VISA

NI-VISA™
Programmer Reference Manual

NATIONAL INSTRUMENTS™

April 1998 Edition
Part Number 321073C-01

Internet Support
E-mail: support@natinst.com
FTP Site: ftp.natinst.com
Web Address: http://www.natinst.com

Bulletin Board Support
BBS United States: 512 794 5422
BBS United Kingdom: 01635 551422
BBS France: 01 48 65 15 59

Fax-on-Demand Support
512 418 1111

Telephone Support (USA)
Tel: 512 795 8248
Fax: 512 794 5678

International Offices
Australia 03 9879 5166, Austria 0662 45 79 90 0, Belgium 02 757 00 20, Brazil 011 288 3336,
Canada (Ontario) 905 785 0085, Canada (Québec) 514 694 8521, Denmark 45 76 26 00, Finland 09 725 725 11,
France 01 48 14 24 24, Germany 089 741 31 30, Hong Kong 2645 3186, Israel 03 6120092, Italy 02 413091,
Japan 03 5472 2970, Korea 02 596 7456, Mexico 5 520 2635, Netherlands 0348 433466, Norway 32 84 84 00,
Singapore 2265886, Spain 91 640 0085, Sweden 08 730 49 70, Switzerland 056 200 51 51, Taiwan 02 377 1200,
United Kingdom 01635 523545

National Instruments Corporate Headquarters
6504 Bridge Point Parkway Austin, Texas 78730-5039 USA Tel: 512 794 0100

Important Information

Warranty

The media on which you receive National Instruments software are warranted not to fail to execute programming instructions, due to defects in materials and workmanship, for a period of 90 days from date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace software media that do not execute programming instructions if National Instruments receives notice of such defects during the warranty period. National Instruments does not warrant that the operation of the software shall be uninterrupted or error free.

A Return Material Authorization (RMA) number must be obtained from the factory and clearly marked on the outside of the package before any equipment will be accepted for warranty work. National Instruments will pay the shipping costs of returning to the owner parts which are covered by warranty.

National Instruments believes that the information in this manual is accurate. The document has been carefully reviewed for technical accuracy. In the event that technical or typographical errors exist, National Instruments reserves the right to make changes to subsequent editions of this document without prior notice to holders of this edition. The reader should consult National Instruments if errors are suspected. In no event shall National Instruments be liable for any damages arising out of or related to this document or the information contained in it.

EXCEPT AS SPECIFIED HEREIN, NATIONAL INSTRUMENTS MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF NATIONAL INSTRUMENTS SHALL BE LIMITED TO THE AMOUNT THERETOFORE PAID BY THE CUSTOMER. NATIONAL INSTRUMENTS WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF. This limitation of the liability of National Instruments will apply regardless of the form of action, whether in contract or tort, including negligence. Any action against National Instruments must be brought within one year after the cause of action accrues. National Instruments shall not be liable for any delay in performance due to causes beyond its reasonable control. The warranty provided herein does not cover damages, defects, malfunctions, or service failures caused by owner's failure to follow the National Instruments installation, operation, or maintenance instructions; owner's modification of the product; owner's abuse, misuse, or negligent acts; and power failure or surges, fire, flood, accident, actions of third parties, or other events outside reasonable control.

Copyright

Trademarks

WARNING REGARDING MEDICAL AND CLINICAL USE OF NATIONAL INSTRUMENTS PRODUCTS

National Instruments products are not designed with components and testing intended to ensure a level of reliability suitable for use in treatment and diagnosis of humans. Applications of National Instruments products involving medical or clinical treatment can create a potential for accidental injury caused by product failure, or by errors on the part of the user or application designer. Any use or application of National Instruments products for or involving medical or clinical treatment must be performed by properly trained and qualified medical personnel, and all traditional medical safeguards, equipment, and procedures that are appropriate in the particular situation to prevent serious injury or death should always continue to be used when National Instruments products are being used. National Instruments products are NOT intended to be a substitute for any form of established process, procedure, or equipment used to monitor or safeguard human health and safety in medical or clinical treatment.

Contents

About This Manual

Chapter 1
Introduction

Chapter 2
Overview of the VISA API

Chapter 3
Attributes

Chapter 4
Events

Chapter 5
Operations

Tables

About This Manual

This manual describes the attributes, events, and operations that comprise the VISA Application Programming Interface (API). This manual is meant to be used with the *NI-VISA User Manual*.

Organization of This Manual

This manual is organized as follows:

- Chapter 1, *Introduction*, lists what you need to get started and presents a brief overview of VISA.
- Chapter 2, *Overview of the VISA API*, contains an overview of the VISA Application Programming Interface (API).
- Chapter 3, *Attributes*, describes the VISA attributes. The attribute descriptions are listed in alphabetical order for easy reference.
- Chapter 4, *Events*, describes the VISA events. The event descriptions are listed in alphabetical order for easy reference.
- Chapter 5, *Operations*, describes the VISA operations. The operation descriptions are listed in alphabetical order for easy reference.
- Appendix A, *Data Types*, lists and describes the type assignments for ANSI C and Visual Basic for each VISA data type.
- Appendix B, *Status Codes*, lists and describes the completion and error codes.
- Appendix C, *Resources*, lists the attributes, events, and operations in each resource in VISA.
- Appendix D, *Customer Communication*, contains forms you can use to request help from National Instruments or to comment on our products and manuals.
- The *Glossary* contains an alphabetical list and description of terms used in this manual, including abbreviations, acronyms, metric prefixes, mnemonics, and symbols.
- The *Index* contains an alphabetical list of key terms and topics in this manual, including the page where you can find each one.

Conventions Used in This Manual

The following conventions are used in this manual:

[ ]	Square brackets are used to denote optional parameters in program code.
☞	This icon to the left of bold italicized text denotes a note, which alerts you to important information.
bold	Bold text denotes parameter names for NI-VISA operations.
bold italic	Bold italic text denotes an note, caution, or warning.
italic	Italic text denotes emphasis, a cross reference, or an introduction to a key concept. This font also denotes text from which you supply the apropriate word or value, as in Windows 3.*x*.
`monospace`	Text in this font denotes text or characters that are to be literally input from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for attributes, events, data types, functions, operations, variables, completion and error codes, and for statements and comments taken from programs.
`monospace bold`	Bold text in this font denotes the messages and responses that the computer automatically prints to the screen. This font also emphasizes lines of code that are different from the other examples.
`monospace italic`	Italic text in this font denotes that you must enter the appropriate words or values in the place of these items.

How to Use This Documentation Set

Use the documentation that came with your GPIB and/or VXI hardware and software kit to install and configure your system.

Refer to the *Read Me First* document for information on installing the NI-VISA distribution media.

Use the *NI-VISA User Manual* for detailed information on how to program using VISA.

Use the *NI-VISA Programmer Reference Manual* for specific information about the attributes, events, and operations, such as format, syntax, parameters, and possible errors.

Related Documentation

The following documents contain information that you may find helpful as you read this manual:

- ANSI/IEEE Standard 488.1-1987, *IEEE Standard Digital Interface for Programmable Instrumentation*
- ANSI/IEEE Standard 488.2-1992, *IEEE Standard Codes, Formats, Protocols, and Common Commands*
- ANSI/IEEE Standard 1014-1987, *IEEE Standard for a Versatile Backplane Bus: VMEbus*
- ANSI/IEEE Standard 1155-1992, *VMEbus Extensions for Instrumentation: VXIbus*
- ANSI/ISO Standard 9899-1990, *Programming Language C*
- *NI-488.2 Function Reference Manual for DOS/Windows*, National Instruments Corporation
- *NI-488.2 User Manual for Windows*, National Instruments Corporation
- *NI-VXI Programmer Reference Manual*, National Instruments Corporation
- *NI-VXI User Manual*, National Instruments Corporation
- VPP-1, *Charter Document*
- VPP-2, *System Frameworks Specification*
- VPP-3.1, *Instrument Drivers Architecture and Design Specification*
- VPP-3.2, *Instrument Driver Developers Specification*
- VPP-3.3, *Instrument Driver Function Panel Specification*
- VPP-4.3, *The VISA Library*
- VPP-4.3.2, *VISA Implementation Specification for Textual Languages*
- VPP-4.3.3, *VISA Implementation Specification for the G Language*
- VPP-5, *VXI Component Knowledge Base Specification*
- VPP-6, *Installation and Packaging Specification*
- VPP-7, *Soft Front Panel Specification*
- VPP-8, *VXI Module/Mainframe to Receiver Interconnection*
- VPP-9, *Instrument Vendor Abbreviations*

Customer Communication

National Instruments wants to receive your comments on our products and manuals. We are interested in the applications you develop with our products, and we want to help if you have problems with them. To make it easy for you to contact us, this manual contains comment and configuration forms for you to complete. These forms are in Appendix D, *Customer Communication*, at the end of this manual.

1

Introduction

This chapter lists what you need to get started and presents a brief overview of VISA.

What You Need to Get Started

- ❑ Appropriate hardware support in the form of a National Instruments GPIB, GPIB-VXI, MXI/VXI or serial interface board. For serial support, the computer's standard serial ports are sufficient.
- ❑ NI-488.2 and/or NI-VXI installed on your system. For serial support, the system's serial drivers are sufficient.
- ❑ NI-VISA distribution media.
- ❑ If you have a GPIB-VXI command module from another vendor, you need that vendor's GPIB-VXI VISA component.

Overview

The VXI*plug&play* Systems Alliance was formed on September 22, 1993 with the goal of increasing end-user success and multivendor interoperability for VXIbus systems. To achieve this goal, VXI*plug&play* defines and implements new levels of standardization to simplify multivendor VXI system integration to benefit both end-users and vendors. As a result, VXI*plug&play* products are easy to use, thanks to new standards for both hardware and software.

At the heart of these standards is the Virtual Instrument Software Architecture, or *VISA*, the I/O software standard on which all VXI*plug&play* software components are based. In the past, there were many different I/O software products to control GPIB and VXI. Software written with these various libraries supplied by individual vendors was directly and uniquely tied to the hardware these vendors produced. The VISA standard, endorsed by over 35 of the largest instrumentation companies in the industry including Tektronix, Hewlett-Packard, and

National Instruments, unifies the industry to make software interoperable, reusable, and able to stand the test of time.

When the VISA standard was initially endorsed, commercial VISA products were not yet available. To quickly realize the benefits of VXI*plug&play*, the alliance developed the VISA Transition Library (VTL) specification. The VTL reflected the alliance's strategy to deliver multi-vendor software interoperability, while at the same time moving the entire industry towards a common, robust VISA foundation for the future. Software written to VTL, such as instrument drivers and executable soft front panels, will also run on present and future VISA implementations without modification.

This manual and the *NI-VISA User Manual* describe how to use NI-VISA, the National Instruments implementation of the VISA I/O standard, in any environment using LabWindows/CVI, any ANSI C compiler, or Microsoft Visual Basic. NI-VISA currently supports the frameworks and programming languages shown in Table 1-1. For information on programming VISA from LabVIEW, refer to the VISA documentation included with your LabVIEW distribution.

Table 1-1. NI-VISA Support

Operating System	Programming Language/ Environment	Framework
Windows 3.*x*	LabWindows/CVI, ANSI C, Visual Basic	WIN
	LabVIEW	GWIN
Windows 95	LabWindows/CVI, ANSI C, Visual Basic	WIN95
	LabVIEW	GWIN95
Windows NT	LabWindows/CVI, ANSI C, Visual Basic	WINNT
	LabVIEW	GWINNT
Solaris 1.*x*	LabWindows/CVI, ANSI C	SUN
Solaris 2.*x*	LabVIEW	GSUN
HP-UX 9	ANSI C, LabWindows/CVI*	HPUX
HP-UX 10	LabVIEW	GHPUX

Table 1-1. NI-VISA Support (Continued)

Operating System	Programming Language/ Environment	Framework
Mac 68K	ANSI C	**
Mac PPC	LabVIEW	**
VxWorks	ANSI C	**

* Although the LabWindows/CVI development environment is not available on HP-UX, the run-time libraries are. Therefore, a LabWindows/CVI application developed on another framework can be ported to HP-UX without modification.

** This framework is not defined by the VXI*plug&play* Systems Alliance, but is still supported by NI-VISA.

The VXI*plug&play* Systems Alliance developed the concept of a *framework* to categorize operating systems, programming languages, and I/O software libraries to bring the most useful products to the most end-users. A framework is a logical grouping of the choices that you face when designing a VXI system. You must always choose an operating system and a programming language along with an application development environment (ADE) when building a system. There are trade-offs associated with each of these decisions; many configurations are possible. The VXI*plug&play* Systems Alliance grouped the most popular operating systems, programming languages, and ADEs into distinct frameworks and defined in-depth specifications to guarantee interoperability of the components within each framework. To claim VXI*plug&play* compliance, a component must be compliant within a given framework.

With this version of NI-VISA, you can perform message-based and register-based communication with instruments, assert triggers, share memory, and respond to interrupts and triggers. You can also perform register accesses at the interface level for VXI. If you find that you need functionality beyond what VISA provides, you can use NI-488.2 or NI-VXI to supplement VISA in the same program. However, it is recommended that you use the VISA API whenever possible.

2

Overview of the VISA API

This chapter contains an overview of the VISA Application Programming Interface (API).

You can use this manual as a reference to the VISA API. This API is partitioned into three distinct mechanisms that access information on a given resource: attributes, events, and operations.

VISA Access Mechanisms

The following paragraphs summarize the most important characteristics of attributes, events, and operations. Please refer to Chapter 3, *VISA Overview*, in the *NI-VISA User Manual* for a more detailed description of this subject.

Attributes

An attribute describes a value within a session or resource that reflects a characteristic of the operational state of the given object. These attributes are accessed through the following operations:

- `viGetAttribute()`
- `viSetAttribute()`

Events

An event is an asynchronous occurrence that is independent of the normal sequential execution of the process running in a system. Depending on how you want to handle event occurrences, you can use the `viEnableEvent()` operation with either the `viInstallHandler()` operation or the `viWaitOnEvent()` operation.

Events respond to attributes in the same manner that resources do. Once your application is done using a particular event received via `viWaitOnEvent()`, it should call `viClose()` to destroy that event.

Operations

An operation is an action defined by a resource that can be performed on the given resource. Each resource has the ability to define a series of operations. In addition to those defined by each resource you can use the following template operations in any resource:

- `viClose()`
- `viGetAttribute()`
- `viSetAttribute()`
- `viStatusDesc()`
- `viTerminate()`
- `viLock()`
- `viUnlock()`
- `viEnableEvent()`
- `viDisableEvent()`
- `viDiscardEvents()`
- `viWaitOnEvent()`
- `viInstallHandler()`
- `viUninstallHandler()`

VISA Resource Types

Currently, there are two VISA resource types—INSTR and MEMACC.

INSTR

A VISA Instrument Control (INSTR) resource lets a controller interact with the device associated with the given resource. Most VISA applications and instrument drivers use only the INSTR resource. This resource type grants the controller the following services to perform message-based and/or register-based I/O, depending on the type of device and the interface to which the device is connected.

Basic I/O services include the ability to send and receive blocks of data to and from the device. The meaning of the data is device dependent, and could be a message, command, or other binary encoded data. For devices compliant with IEEE-488, the basic I/O services also include triggering (both software and hardware), servicing requests, reading status bytes, and clearing the device.

Formatted I/O services provide both formatted and buffered I/O capabilities for data transfers to and from devices. The formatting capabilities include those specified by ANSI C, with extensions for common protocols used by instrumentation systems. Buffering improves system performance by making it possible to not only transfer large blocks of data, but also send several commands at one time.

Memory I/O (or Register I/O) services allow register-level access to devices connected to interfaces that support direct memory access, such as the VXIbus or VMEbus. Both high-level and low-level access services have operations for individual register accesses, with a trade-off between speed and complexity. The high-level access services also have operations for moving large blocks of data to and from devices. When using an INSTR resource, all address parameters are relative to the device's assigned memory base in the given address space; knowing a device's base address is neither required by nor relevant to the user.

Shared Memory services make it possible to allocate memory on a particular device that is to be used exclusively by a given session. This is usually available only on devices that export shared memory specifically for such a purpose, such as a VXIbus or VMEbus controller.

MEMACC

A VISA Memory Access (MEMACC) resource lets a controller interact with the interface associated with the given resource. Advanced users who need to perform memory accesses directly between multiple devices typically use the MEMACC resource. This resource type gives the controller the following services to access arbitrary registers or memory addresses on memory-mapped buses.

Memory I/O (or Register I/O) services allow register level access to interfaces that support direct memory access, such as the VXIbus or VMEbus. Both high-level and low-level access services have operations for individual register accesses, with a trade-off between speed and complexity. The high-level access services also have operations for moving large blocks of data to and from arbitrary addresses. When using a MEMACC resource, all address parameters are absolute within the given address space; knowing a device's base address is both required by and relevant to the user.

Description of the API

The following three chapters describe the individual attributes, events, and operations. These are listed in alphabetical order within each access mechanism. Since a particular item can refer to more than one resource or interface type, each item is clearly marked with the resource and interface that support it.

Refer to Appendix C, *Resources*, for a quick reference of how the attributes, events, and operations map to the available resources.

3

Attributes

This chapter describes the VISA attributes. The attribute descriptions are listed in alphabetical order for easy reference.

Each attribute description contains a checkbox table below the title indicating the supported interface(s), whether Serial, GPIB, GPIB-VXI, and/or VXI; the checkbox is filled in for those that are applicable. The Attribute Information table lists the access privilege, the data type, range of values, and the default value.

VI_ATTR_ASRL_AVAIL_NUM

■ Serial	❑ GPIB	❑ GPIB-VXI	❑ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	`ViUInt32`	0 to FFFFFFFFh	N/A

Description

`VI_ATTR_ASRL_AVAIL_NUM` shows the number of bytes available in the global receive buffer.

Related Items

See the *INSTR Resource* description in Appendix C, *Resources*.

VI_ATTR_ASRL_BAUD

■ Serial	❑ GPIB	❑ GPIB-VXI	❑ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Global	ViUInt32	0 to FFFFFFFFh	9600

Description

VI_ATTR_ASRL_BAUD is the baud rate of the interface. It is represented as an unsigned 32-bit integer so that any baud rate can be used, but it usually requires a commonly used rate such as 300, 1200, 2400, or 9600 baud.

Related Items

See the VI_ATTR_ASRL_DATA_BITS, VI_ATTR_ASRL_FLOW_CNTRL, VI_ATTR_ASRL_PARITY, and VI_ATTR_ASRL_STOP_BITS descriptions in this chapter. Also see the *INSTR Resource* description in Appendix C, *Resources*.

VI_ATTR_ASRL_CTS_STATE

■ Serial	❑ GPIB	❑ GPIB-VXI	❑ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViInt16	VI_STATE_ASSERTED (1) VI_STATE_UNASSERTED (0) VI_STATE_UNKNOWN (–1)	N/A

Description

VI_ATTR_ASRL_CTS_STATE shows the current state of the Clear To Send (CTS) input signal.

Related Items

See the VI_ATTR_ASRL_DCD_STATE, VI_ATTR_ASRL_DSR_STATE, VI_ATTR_ASRL_DTR_STATE, VI_ATTR_ASRL_RI_STATE, and VI_ATTR_ASRL_RTS_STATE descriptions in this chapter. Also see the *INSTR Resource* description in Appendix C, *Resources*.

VI_ATTR_ASRL_DATA_BITS

■ Serial	❑ GPIB	❑ GPIB-VXI	❑ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Global	ViUInt16	5 to 8	8

Description

VI_ATTR_ASRL_DATA_BITS is the number of data bits contained in each frame (from 5 to 8). The data bits for each frame are located in the low-order bits of every byte stored in memory.

Related Items

See the VI_ATTR_ASRL_BAUD, VI_ATTR_ASRL_FLOW_CNTRL, VI_ATTR_ASRL_PARITY, and VI_ATTR_ASRL_STOP_BITS descriptions in this chapter. Also see the *INSTR Resource* description in Appendix C, *Resources*.

VI_ATTR_ASRL_DCD_STATE

■ Serial	❑ GPIB	❑ GPIB-VXI	❑ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViInt16	VI_STATE_ASSERTED (1) VI_STATE_UNASSERTED (0) VI_STATE_UNKNOWN (–1)	N/A

Description

VI_ATTR_ASRL_DCD_STATE shows the current state of the Data Carrier Detect (DCD) input signal. The DCD signal is often used by modems to indicate the detection of a carrier (remote modem) on the telephone line. The DCD signal is also known as *Receive Line Signal Detect* (RLSD).

Related Items

See the VI_ATTR_ASRL_CTS_STATE, VI_ATTR_ASRL_DSR_STATE, VI_ATTR_ASRL_DTR_STATE, VI_ATTR_ASRL_RI_STATE, and VI_ATTR_ASRL_RTS_STATE descriptions in this chapter. Also see the *INSTR Resource* description in Appendix C, *Resources*.

VI_ATTR_ASRL_DSR_STATE

■ Serial	❑ GPIB	❑ GPIB-VXI	❑ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViInt16	VI_STATE_ASSERTED (1) VI_STATE_UNASSERTED (0) VI_STATE_UNKNOWN (–1)	N/A

Description

VI_ATTR_ASRL_DSR_STATE shows the current state of the Data Set Ready (DSR) input signal.

Related Items

See the VI_ATTR_ASRL_CTS_STATE, VI_ATTR_ASRL_DCD_STATE, VI_ATTR_ASRL_DTR_STATE, VI_ATTR_ASRL_RI_STATE, and VI_ATTR_ASRL_RTS_STATE descriptions in this chapter. Also see the *INSTR Resource* description in Appendix C, *Resources*.

VI_ATTR_ASRL_DTR_STATE

■ Serial	❑ GPIB	❑ GPIB-VXI	❑ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Global	ViInt16	VI_STATE_ASSERTED (1) VI_STATE_UNASSERTED (0) VI_STATE_UNKNOWN (–1)	N/A

Description

VI_ATTR_ASRL_DTR_STATE shows the current state of the Data Terminal Ready (DTR) input signal. When the VI_ATTR_ASRL_FLOW_CNTRL attribute is set to VI_ASRL_FLOW_DTR_DSR, this attribute is ignored when changed, but can be read to determine whether the background flow control is asserting or unasserting the signal.

Related Items

See the VI_ATTR_ASRL_CTS_STATE, VI_ATTR_ASRL_DCD_STATE, VI_ATTR_ASRL_DSR_STATE, VI_ATTR_ASRL_RI_STATE, and VI_ATTR_ASRL_RTS_STATE descriptions in this chapter. Also see the *INSTR Resource* description in Appendix C, *Resources*.

VI_ATTR_ASRL_END_IN

■ Serial	❑ GPIB	❑ GPIB-VXI	❑ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt16	VI_ASRL_END_NONE (0) VI_ASRL_END_LAST_BIT (1) VI_ASRL_END_TERMCHAR (2)	VI_ASRL_END_TERMCHAR

Description

VI_ATTR_ASRL_END_IN indicates the method used to terminate read operations.

- If it is set to VI_ASRL_END_NONE, the read will not terminate until all of the requested data is received (or an error occurs).
- If it is set to VI_ASRL_END_LAST_BIT, the read will terminate as soon as a character arrives with its last bit set. For example, if VI_ATTR_ASRL_DATA_BITS is set to 8, the read will terminate when a character arrives with the 8th bit set.
- If it is set to VI_ASRL_END_TERMCHAR, the read will terminate as soon as the character in VI_ATTR_TERMCHAR is received. In this case, VI_ATTR_TERMCHAR_EN is ignored.

Because the default value of VI_ATTR_TERMCHAR is 0Ah (linefeed), read operations on serial ports will stop reading whenever a linefeed is encountered. To change this behavior, you must change the value of one of these attributes—VI_ATTR_ASRL_END_IN or VI_ATTR_TERMCHAR.

Related Items

See the VI_ATTR_ASRL_END_OUT and VI_ATTR_TERMCHAR descriptions in this chapter. Also see the *INSTR Resource* description in Appendix C, *Resources*.

VI_ATTR_ASRL_END_OUT

■ Serial	❑ GPIB	❑ GPIB-VXI	❑ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt16	VI_ASRL_END_NONE (0) VI_ASRL_END_LAST_BIT (1) VI_ASRL_END_TERMCHAR (2) VI_ASRL_END_BREAK (3)	VI_ASRL_END_NONE

Description

VI_ATTR_ASRL_END_OUT indicates the method used to terminate write operations.

- If it is set to VI_ASRL_END_NONE, the write will not append anything to the data being written.
- If it is set to VI_ASRL_END_LAST_BIT, the write will send all but the last character with the last bit clear, then transmit the last character with the last bit set. For example, if VI_ATTR_ASRL_DATA_BITS is set to 8, the write will clear the 8th bit for all but the last character, then transmit the last character with the 8th bit set.
- If it is set to VI_ASRL_END_TERMCHAR, the write will send the character in VI_ATTR_TERMCHAR after the data being transmitted.
- If it is set to VI_ASRL_END_BREAK, the write will transmit a break after all the characters for the write have been sent.

Related Items

See the VI_ATTR_ASRL_END_IN and VI_ATTR_TERMCHAR descriptions in this chapter. Also see the *INSTR Resource* description in Appendix C, *Resources*.

VI_ATTR_ASRL_FLOW_CNTRL

■ Serial	❑ GPIB	❑ GPIB-VXI	❑ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Global	ViUInt16	VI_ASRL_FLOW_NONE (0) VI_ASRL_FLOW_XON_XOFF (1) VI_ASRL_FLOW_RTS_CTS (2) VI_ASRL_FLOW_DTR_DSR (4)	VI_ASRL_FLOW_NONE

Description

VI_ATTR_ASRL_FLOW_CNTRL indicates the type of flow control used by the transfer mechanism.

- If this attribute is set to VI_ASRL_FLOW_NONE, the transfer mechanism does not use flow control, and buffers on both sides of the connection are assumed to be large enough to hold all data transferred.
- If this attribute is set to VI_ASRL_FLOW_XON_XOFF, the transfer mechanism uses the XON and XOFF characters to perform flow control. The transfer mechanism controls input flow by sending XOFF when the receive buffer is nearly full, and it controls the output flow by suspending transmission when XOFF is received.
- If this attribute is set to VI_ASRL_FLOW_RTS_CTS, the transfer mechanism uses the RTS output signal and the CTS input signal to perform flow control. The transfer mechanism controls input flow by unasserting the RTS signal when the receive buffer is nearly full, and it controls output flow by suspending the transmission when the CTS signal is unasserted.
- If this attribute is set to VI_ASRL_FLOW_DTR_DSR, the transfer mechanism uses the DTR output signal and the DSR input signal to perform flow control. The transfer mechanism controls input flow by unasserting the DTR signal when the receive buffer is nearly full, and it controls output flow by suspending the transmission when the DSR signal is unasserted.

This attribute can specify multiple flow control mechanisms by bit-ORing multiple values together. However, certain combinations may not be supported by all serial ports and/or operating systems.

Related Items

See the VI_ATTR_ASRL_BAUD, VI_ATTR_ASRL_DATA_BITS, VI_ATTR_ASRL_PARITY, and VI_ATTR_ASRL_STOP_BITS descriptions in this chapter. Also see the *INSTR Resource* description in Appendix C, *Resources*.

VI_ATTR_ASRL_PARITY

■ Serial	❑ GPIB	❑ GPIB-VXI	❑ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Global	ViUInt16	VI_ASRL_PAR_NONE (0) VI_ASRL_PAR_ODD (1) VI_ASRL_PAR_EVEN (2) VI_ASRL_PAR_MARK (3) VI_ASRL_PAR_SPACE (4)	VI_ASRL_PAR_NONE

Description

VI_ATTR_ASRL_PARITY is the parity used with every frame transmitted and received.

- VI_ASRL_PAR_MARK means that the parity bit exists and is always 1.
- VI_ASRL_PAR_SPACE means that the parity bit exists and is always 0.

Related Items

See the VI_ATTR_ASRL_BAUD, VI_ATTR_ASRL_DATA_BITS, VI_ATTR_ASRL_FLOW_CNTRL, and VI_ATTR_ASRL_STOP_BITS descriptions in this chapter. Also see the *INSTR Resource* description in Appendix C, *Resources*.

VI_ATTR_ASRL_REPLACE_CHAR

■ Serial	❑ GPIB	❑ GPIB-VXI	❑ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt8	0 to FFh	0

Description

VI_ATTR_ASRL_REPLACE_CHAR specifies the character to be used to replace incoming characters that arrive with errors (such as parity error).

Related Items

See the *INSTR Resource* description in Appendix C, *Resources*.

VI_ATTR_ASRL_RI_STATE

■ Serial	❑ GPIB	❑ GPIB-VXI	❑ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViInt16	VI_STATE_ASSERTED (1) VI_STATE_UNASSERTED (0) VI_STATE_UNKNOWN (–1)	N/A

Description

VI_ATTR_ASRL_RI_STATE shows the current state of the Ring Indicator (RI) input signal. The RI signal is often used by modems to indicate that the telephone line is ringing.

Related Items

See the VI_ATTR_ASRL_CTS_STATE, VI_ATTR_ASRL_DCD_STATE, VI_ATTR_ASRL_DSR_STATE, VI_ATTR_ASRL_DTR_STATE, and VI_ATTR_ASRL_RTS_STATE descriptions in this chapter. Also see the *INSTR Resource* description in Appendix C, *Resources*.

VI_ATTR_ASRL_RTS_STATE

■ Serial	❑ GPIB	❑ GPIB-VXI	❑ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Global	ViInt16	VI_STATE_ASSERTED (1) VI_STATE_UNASSERTED (0) VI_STATE_UNKNOWN (–1)	N/A

Description

VI_ATTR_ASRL_RTS_STATE is used to manually assert or unassert the Request To Send (RTS) output signal. When the VI_ATTR_ASRL_FLOW_CNTRL attribute is set to VI_ASRL_FLOW_RTS_CTS, this attribute is ignored when changed, but can be read to determine whether the background flow control is asserting or unasserting the signal.

Related Items

See the VI_ATTR_ASRL_CTS_STATE, VI_ATTR_ASRL_DCD_STATE, VI_ATTR_ASRL_DSR_STATE, VI_ATTR_ASRL_DTR_STATE, and VI_ATTR_ASRL_RI_STATE descriptions in this chapter. Also see the *INSTR Resource* description in Appendix C, *Resources*.

VI_ATTR_ASRL_STOP_BITS

■ Serial	❑ GPIB	❑ GPIB-VXI	❑ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Global	ViUInt16	VI_ASRL_STOP_ONE (10) VI_ASRL_STOP_ONE5 (15) VI_ASRL_STOP_TWO (20)	VI_ASRL_STOP_ONE

Description

VI_ATTR_ASRL_STOP_BITS is the number of stop bits used to indicate the end of a frame. The value VI_ASRL_STOP_ONE5 indicates one-and-one-half (1.5) stop bits.

Related Items

See the VI_ATTR_ASRL_BAUD, VI_ATTR_ASRL_DATA_BITS, VI_ATTR_ASRL_FLOW_CNTRL, and VI_ATTR_ASRL_PARITY descriptions in this chapter. Also see the *INSTR Resource* description in Appendix C, *Resources*.

VI_ATTR_ASRL_XOFF_CHAR

■ Serial	❑ GPIB	❑ GPIB-VXI	❑ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt8	0 to FFh	<Control-S> (13h)

Description

VI_ATTR_ASRL_XOFF_CHAR specifies the value of the XOFF character used for XON/XOFF flow control (both directions). If XON/XOFF flow control (software handshaking) is not being used, the value of this attribute is ignored.

Related Items

See the VI_ATTR_ASRL_XON_CHAR and VI_ATTR_ASRL_FLOW_CNTRL descriptions in this chapter. Also see the *INSTR Resource* description in Appendix C, *Resources*.

VI_ATTR_ASRL_XON_CHAR

■ Serial	❑ GPIB	❑ GPIB-VXI	❑ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt8	0 to FFh	<Control-Q> (11h)

Description

VI_ATTR_ASRL_XON_CHAR specifies the value of the XON character used for XON/XOFF flow control (both directions). If XON/XOFF flow control (software handshaking) is not being used, the value of this attribute is ignored.

Related Items

See the VI_ATTR_ASRL_XOFF_CHAR and VI_ATTR_ASRL_FLOW_CNTRL descriptions in this chapter. Also see the *INSTR Resource* description in Appendix C, *Resources*.

VI_ATTR_BUFFER

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only	ViBuf	N/A	N/A

Description

VI_ATTR_BUFFER contains the address of a buffer that was used in an asynchronous operation.

Related Items

See the VI_ATTR_STATUS, VI_ATTR_JOB_ID, and VI_ATTR_RET_COUNT descriptions in this chapter. See the VI_EVENT_IO_COMPLETION event description in Chapter 4, *Events*. Also see the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix C, *Resources*.

VI_ATTR_CMDR_LA

❑ Serial	❑ GPIB	■ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViInt16	0 to 255 VI_UNKNOWN_LA (–1)	N/A

Description

VI_ATTR_CMDR_LA is the unique logical address of the commander of the VXI device used by the given session.

Related Items

See the *INSTR Resource* description in Appendix C, *Resources*.

VI_ATTR_DEST_ACCESS_PRIV

❑ Serial	❑ GPIB	■ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt16	VI_DATA_PRIV (0) VI_DATA_NPRIV (1) VI_PROG_PRIV (2) VI_PROG_NPRIV (3) VI_BLCK_PRIV (4) VI_BLCK_NPRIV (5) VI_D64_PRIV (6) VI_D64_NPRIV (7)	VI_DATA_PRIV

Description

VI_ATTR_DEST_ACCESS_PRIV specifies the address modifier to be used in high-level access operations, such as viOut*XX*( ) and viMoveOut*XX*( ), when writing to the destination.

Related Items

See the VI_ATTR_DEST_BYTE_ORDER, VI_ATTR_DEST_INCREMENT, VI_ATTR_SRC_ACCESS_PRIV, and VI_ATTR_WIN_ACCESS_PRIV descriptions in this chapter. Also see the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix C, *Resources*.

VI_ATTR_DEST_BYTE_ORDER

❑ Serial	❑ GPIB	■ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt16	VI_BIG_ENDIAN (0) VI_LITTLE_ENDIAN (1)	VI_BIG_ENDIAN

Description

VI_ATTR_DEST_BYTE_ORDER specifies the byte order to be used in high-level access operations, such as viOut*XX*( ) and viMoveOut*XX*( ), when writing to the destination.

Related Items

See the VI_ATTR_DEST_ACCESS_PRIV, VI_ATTR_DEST_INCREMENT, VI_ATTR_SRC_BYTE_ORDER, and VI_ATTR_WIN_BYTE_ORDER descriptions in this chapter. Also see the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix C, *Resources*.

VI_ATTR_DEST_INCREMENT

❑ Serial	❑ GPIB	■ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViInt32	0 to 1	1

Description

VI_ATTR_DEST_INCREMENT is used in the viMoveOut*XX*( ) operations to specify by how many elements the destination offset is to be incremented after every transfer. The default value of this attribute is 1 (that is, the destination address will be incremented by 1 after each transfer), and the viMoveOut*XX*( ) operations move into consecutive elements. If this attribute is set to 0, the viMoveOut*XX*( ) operations will always write to the same element, essentially treating the destination as a FIFO register.

Related Items

See the VI_ATTR_DEST_ACCESS_PRIV, VI_ATTR_DEST_BYTE_ORDER, and VI_ATTR_SRC_INCREMENT descriptions in this chapter. Also see the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix C, *Resources*.

VI_ATTR_EVENT_TYPE

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only	ViEventType	0h to FFFFFFFFh	N/A

Description

VI_ATTR_EVENT_TYPE is the unique logical identifier for the event type of the specified event.

Related Items

Refer to Chapter 4, *Events*, for a list of events.

VI_ATTR_FDC_CHNL

❑ Serial	❑ GPIB	■ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt16	0 to 7	N/A

Description

VI_ATTR_FDC_CHNL determines which Fast Data Channel (FDC) will be used to transfer the buffer.

Related Items

See the VI_ATTR_FDC_GEN_SIGNAL_EN, VI_ATTR_FDC_MODE, and VI_ATTR_FDC_USE_PAIR descriptions in this chapter. Also see the *INSTR Resource* description in Appendix C, *Resources*.

VI_ATTR_FDC_GEN_SIGNAL_EN

❑ Serial	❑ GPIB	■ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViBoolean	VI_TRUE (1) VI_FALSE (0)	VI_FALSE

Description

Setting VI_ATTR_FDC_GEN_SIGNAL_EN to VI_TRUE lets the servant send a signal when control of the FDC channel is passed back to the commander. This action frees the commander from having to poll the FDC header while engaging in an FDC transfer.

Related Items

See the VI_ATTR_FDC_CHNL, VI_ATTR_FDC_MODE, and VI_ATTR_FDC_USE_PAIR descriptions in this chapter. Also see the *INSTR Resource* description in Appendix C, *Resources*.

VI_ATTR_FDC_MODE

❑ Serial	❑ GPIB	■ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt16	VI_FDC_NORMAL (1) VI_FDC_STREAM (2)	VI_FDC_NORMAL

Description

VI_ATTR_FDC_MODE specifies which Fast Data Channel (FDC) mode to use (either normal or stream mode).

Related Items

See the VI_ATTR_FDC_CHNL, VI_ATTR_FDC_GEN_SIGNAL_EN, and VI_ATTR_FDC_USE_PAIR descriptions in this chapter. Also see the *INSTR Resource* description in Appendix C, *Resources*.

VI_ATTR_FDC_USE_PAIR

❑ Serial	❑ GPIB	■ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViBoolean	VI_TRUE (1) VI_FALSE (0)	VI_FALSE

Description

Setting VI_ATTR_FDC_USE_PAIR to VI_TRUE specifies to use a channel pair for transferring data. Otherwise, only one channel will be used.

Related Items

See the VI_ATTR_FDC_CHNL, VI_ATTR_FDC_GEN_SIGNAL_EN, and VI_ATTR_FDC_MODE descriptions in this chapter. Also see the *INSTR Resource* description in Appendix C, *Resources*.

VI_ATTR_GPIB_PRIMARY_ADDR

❑ Serial	■ GPIB	■ GPIB-VXI	❑ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViUInt16	0 to 30	N/A

Description

VI_ATTR_GPIB_PRIMARY_ADDR specifies the primary address of the GPIB device used by the given session.

Related Items

See the VI_ATTR_GPIB_SECONDARY_ADDR description in this chapter. Also see the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix C, *Resources*.

VI_ATTR_GPIB_READDR_EN

❑ Serial	■ GPIB	■ GPIB-VXI	❑ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViBoolean	VI_TRUE (1) VI_FALSE (0)	VI_TRUE

Description

VI_ATTR_GPIB_READDR_EN specifies whether to use repeat addressing before each read or write operation.

Related Items

See the VI_ATTR_GPIB_UNADDR_EN description in this chapter. Also see the *INSTR Resource* description in Appendix C, *Resources*.

VI_ATTR_GPIB_REN_STATE

❑ Serial	■ GPIB	■ GPIB-VXI	❑ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViBoolean	VI_TRUE (1) VI_FALSE (0)	N/A

Description

VI_ATTR_GPIB_REN_STATE returns the current state of the GPIB REN interface line.

Related Items

See the *INSTR Resource* description in Appendix C, *Resources*.

VI_ATTR_GPIB_SECONDARY_ADDR

❑ Serial	■ GPIB	■ GPIB-VXI	❑ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViUInt16	0 to 30, VI_NO_SEC_ADDR (FFFFh)	N/A

Description

VI_ATTR_GPIB_SECONDARY_ADDR specifies the secondary address of the GPIB device used by the given session.

Related Items

See the VI_ATTR_GPIB_PRIMARY_ADDR description in this chapter. Also see the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix C, *Resources*.

VI_ATTR_GPIB_UNADDR_EN

❑ Serial	■ GPIB	■ GPIB-VXI	❑ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViBoolean	VI_TRUE (1) VI_FALSE (0)	VI_FALSE

Description

VI_ATTR_GPIB_UNADDR_EN specifies whether to unaddress the device (UNT and UNL) after each read or write operation.

Related Items

See the VI_ATTR_GPIB_READDR_EN description in this chapter. Also see the *INSTR Resource* description in Appendix C, *Resources*.

VI_ATTR_IMMEDIATE_SERV

❑ Serial	❑ GPIB	■ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViBoolean	VI_TRUE (1) VI_FALSE (0)	VI_FALSE

Description

VI_ATTR_IMMEDIATE_SERV specifies whether the device associated with this session is an immediate servant of the controller running VISA.

Related Items

See the *INSTR Resource* description in Appendix C, *Resources*.

VI_ATTR_INTF_INST_NAME

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	`ViString`	N/A	N/A

Description

`VI_ATTR_INTF_INST_NAME` specifies human-readable text that describes the given interface.

Related Items

See the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix C, *Resources*.

VI_ATTR_INTF_NUM

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViUInt16	0h to FFFFh	0

Description

VI_ATTR_INTF_NUM specifies the board number for the given interface.

Related Items

See the VI_ATTR_INTF_TYPE description in this chapter. Also see the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix C, *Resources*.

VI_ATTR_INTF_PARENT_NUM

❑ Serial	❑ GPIB	■ GPIB-VXI	❑ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViUInt16	0h to FFFFh	0

Description

VI_ATTR_INTF_PARENT_NUM specifies the board number of the GPIB board to which the GPIB-VXI is attached.

Related Items

See the VI_ATTR_INTF_NUM description in this chapter. Also see the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix C, *Resources*.

VI_ATTR_INTF_TYPE

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViUInt16	VI_INTF_ASRL (4) VI_INTF_GPIB (1) VI_INTF_GPIB_VXI (3) VI_INTF_VXI (2)	N/A

Description

VI_ATTR_INTF_TYPE specifies the interface type of the given session.

Related Items

See the VI_ATTR_INTF_NUM description in this chapter. Also see the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix C, *Resources*.

VI_ATTR_INTR_STATUS_ID

❑ Serial	❑ GPIB	❑ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViUInt32	0 to FFFFFFFFh	N/A

Description

VI_ATTR_INTR_STATUS_ID specifies the 32-bit status/ID retrieved during the IACK cycle.

Related Items

See the VI_ATTR_EVENT_TYPE and VI_ATTR_RECV_INTR_LEVEL descriptions in this chapter. See the VI_EVENT_VXI_VME_INTR event description in Chapter 4, *Events*. Also see the *INSTR Resource* description in Appendix C, *Resources*.

VI_ATTR_IO_PROT

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt16	**GPIB**: VI_NORMAL (1) VI_HS488 (3)	VI_NORMAL
		VXI, **GPIB-VXI**: VI_NORMAL (1) VI_FDC (2)	VI_NORMAL
		Serial: VI_NORMAL (1) VI_ASRL488 (4)	VI_NORMAL

Description

VI_ATTR_IO_PROT specifies which protocol to use. In VXI systems, for example, you can choose between normal word serial or Fast Data Channel (FDC). In GPIB, you can choose between normal and high-speed (HS488) data transfers. In serial systems, you can choose between normal and 488-style transfers.

Related Items

See the VI_ATTR_FDC_CHNL, VI_ATTR_FDC_GEN_SIGNAL_EN, VI_ATTR_FDC_MODE, and VI_ATTR_FDC_USE_PAIR descriptions in this chapter. Also see the *INSTR Resource* description in Appendix C, *Resources*.

VI_ATTR_JOB_ID

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only	ViJobId	N/A	N/A

Description

VI_ATTR_JOB_ID contains the job ID of the asynchronous operation that has completed.

Related Items

See the VI_ATTR_STATUS, VI_ATTR_BUFFER, and VI_ATTR_RET_COUNT descriptions in this chapter. See the VI_EVENT_IO_COMPLETION event description in Chapter 4, *Events*. Also see the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix C, *Resources*.

VI_ATTR_MAINFRAME_LA

❑ Serial	❑ GPIB	■ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViInt16	0 to 255 VI_UNKNOWN_LA (–1)	N/A

Description

VI_ATTR_MAINFRAME_LA specifies the lowest logical address in the mainframe. If the logical address is not known, VI_UNKNOWN_LA is returned.

Related Items

See the *INSTR Resource* description in Appendix C, *Resources*.

VI_ATTR_MANF_ID

❑ Serial	❑ GPIB	■ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViUInt16	0h to FFFh	N/A

Description

VI_ATTR_MANF_ID is the manufacturer identification number of the VXIbus device.

Related Items

See the VI_ATTR_MODEL_CODE description in this chapter. Also see the *INSTR Resource* description in Appendix C, *Resources*.

VI_ATTR_MAX_QUEUE_LENGTH

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt32	1h to FFFFFFFFh	50

Description

VI_ATTR_MAX_QUEUE_LENGTH specifies the maximum number of events that can be queued at any time on the given session. Events that occur after the queue has become full will be discarded.

VI_ATTR_MAX_QUEUE_LENGTH is a Read/Write attribute until the first time viEnableEvent() is called on a session. Thereafter, this attribute is Read Only.

Related Items

See the viEnableEvent() and viWaitOnEvent() descriptions in Chapter 5, *Operations*. Also see the *VISA Resource Template* description in Appendix C, *Resources*.

VI_ATTR_MEM_BASE

❑ Serial	❑ GPIB	■ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViBusAddress	0h to FFFFFFFFh	N/A

Description

VI_ATTR_MEM_BASE specifies the base address of the device in VXIbus memory address space. This base address is applicable to A24 or A32 address space. If the value of VI_ATTR_MEM_SPACE is VI_A16_SPACE, the value of this attribute is meaningless for VXI devices.

Related Items

See the VI_ATTR_MEM_SIZE and VI_ATTR_MEM_SPACE descriptions in this chapter. Also see the *INSTR Resource* description in Appendix C, *Resources*.

VI_ATTR_MEM_SIZE

❑ Serial	❑ GPIB	■ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViBusSize	0h to FFFFFFFFh	N/A

Description

VI_ATTR_MEM_SIZE specifies the size of memory requested by the device in VXIbus address space. If the value of VI_ATTR_MEM_SPACE is VI_A16_SPACE, the value of this attribute is meaningless for VXI devices.

Related Items

See the VI_ATTR_MEM_BASE and VI_ATTR_MEM_SPACE descriptions in this chapter. Also see the *INSTR Resource* description in Appendix C, *Resources*.

VI_ATTR_MEM_SPACE

❑ Serial	❑ GPIB	■ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViUInt16	VI_A16_SPACE (1) VI_A24_SPACE (2) VI_A32_SPACE (3)	VI_A16_SPACE

Description

VI_ATTR_MEM_SPACE specifies the VXIbus address space used by the device. The three types are A16, A24, or A32 memory address space.

A VXI device with memory in A24 or A32 space also has registers accessible in the configuration section of A16 space. A VME device with memory in multiple address spaces requires one VISA resource for each address space used.

Related Items

See the VI_ATTR_MEM_BASE and VI_ATTR_MEM_SIZE descriptions in this chapter. Also see the *INSTR Resource* description in Appendix C, *Resources*.

VI_ATTR_MODEL_CODE

❑ Serial	❑ GPIB	■ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViUInt16	0h to FFFFh	N/A

Description

VI_ATTR_MODEL_CODE specifies the model code for the VXIbus device.

Related Items

See the VI_ATTR_MANF_ID description in this chapter. Also see the *INSTR Resource* description in Appendix C, *Resources*.

VI_ATTR_OPER_NAME

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only	ViString	N/A	N/A

Description

VI_ATTR_OPER_NAME contains the name of the operation generating this event.

Related Items

See the VI_ATTR_STATUS and VI_ATTR_EVENT_TYPE descriptions in this chapter. See the VI_EVENT_EXCEPTION and VI_EVENT_IO_COMPLETION event descriptions in Chapter 4, *Events*. Also see the *INSTR Resource* description in Appendix C, *Resources*.

VI_ATTR_RD_BUF_OPER_MODE

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt16	VI_FLUSH_ON_ACCESS (1) VI_FLUSH_DISABLE (3)	VI_FLUSH_DISABLE

Description

VI_ATTR_RD_BUF_OPER_MODE specifies the operational mode of the formatted I/O read buffer. When the operational mode is set to VI_FLUSH_DISABLE (default), the buffer is flushed only on explicit calls to viFlush(). If the operational mode is set to VI_FLUSH_ON_ACCESS, the write buffer is flushed every time a viScanf() (or related) operation completes.

Related Items

See the VI_ATTR_WR_BUF_OPER_MODE description in this chapter. See the viFlush() and viScanf() descriptions in Chapter 5, *Operations*. Also see the *INSTR Resource* description in Appendix C, *Resources*.

VI_ATTR_RECV_INTR_LEVEL

❑ Serial	❑ GPIB	❑ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only	ViInt16	1 to 7; VI_UNKNOWN_LEVEL (–1)	N/A

Description

VI_ATTR_RECV_INTR_LEVEL is the VXI interrupt level on which the interrupt was received.

Related Items

See the VI_ATTR_EVENT_TYPE and VI_ATTR_INTR_STATUS_ID descriptions in this chapter. See the VI_EVENT_VXI_VME_INTR event description in Chapter 4, *Events*. Also see the *INSTR Resource* description in Appendix C, *Resources*.

VI_ATTR_RECV_TRIG_ID

❑ Serial	❑ GPIB	❑ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only	ViInt16	VI_TRIG_TTL0 (0) to VI_TRIG_TTL7 (7); VI_TRIG_ECL0 (8) to VI_TRIG_ECL1 (9)	N/A

Description

VI_ATTR_RECV_TRIG_ID identifies the triggering mechanism on which the specified trigger event was received.

Related Items

See the VI_EVENT_TRIG event description in Chapter 4, *Events*. Also see the *INSTR Resource* description in Appendix C, *Resources*.

VI_ATTR_RET_COUNT

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only	ViUInt32	0h to FFFFFFFFh	N/A

Description

VI_ATTR_RET_COUNT contains the actual number of elements that were asynchronously transferred.

Related Items

See the VI_ATTR_STATUS, VI_ATTR_JOB_ID, and VI_ATTR_BUFFER descriptions in this chapter. See the VI_EVENT_IO_COMPLETION event description in Chapter 4, *Events*. Also see the *INSTR Resource* description in Appendix C, *Resources*.

VI_ATTR_RM_SESSION

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Local	ViSession	N/A	N/A

Description

VI_ATTR_RM_SESSION specifies the session of the Resource Manager that was used to open this session.

Related Items

See the *VISA Resource Template* description in Appendix C, *Resources*.

VI_ATTR_RSRC_IMPL_VERSION

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViVersion	0h to FFFFFFFFh	N/A

Description

VI_ATTR_RSRC_IMPL_VERSION is the resource version that uniquely identifies each of the different revisions or implementations of a resource. This attribute value is defined by the individual manufacturer and increments with each new revision. The format of the value has the upper 12 bits as the major number of the version, the next lower 12 bits as the minor number of the version, and the lowest 8 bits as the sub-minor number of the version.

Related Items

See the VI_ATTR_RSRC_SPEC_VERSION description in this chapter. Also see the *VISA Resource Template* description in Appendix C, *Resources*.

VI_ATTR_RSRC_LOCK_STATE

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViAccessMode	VI_NO_LOCK (0) VI_EXCLUSIVE_LOCK (1) VI_SHARED_LOCK (2)	VI_NO_LOCK

Description

VI_ATTR_RSRC_LOCK_STATE indicates the current locking state of the resource. The resource can be unlocked, locked with an exclusive lock, or locked with a shared lock.

Related Items

See the *VISA Resource Template* description in Appendix C, *Resources*.

VI_ATTR_RSRC_MANF_ID

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViUInt16	0h to 3FFFh	N/A

Description

VI_ATTR_RSRC_MANF_ID is a value that corresponds to the VXI manufacturer ID of the vendor that implemented the VISA library.

Related Items

See the VI_ATTR_RSRC_MANF_NAME description in this chapter. Also see the *VISA Resource Template* description in Appendix C, *Resources*.

VI_ATTR_RSRC_MANF_NAME

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViString	N/A	N/A

Description

VI_ATTR_RSRC_MANF_NAME is a string that corresponds to the manufacturer name of the vendor that implemented the VISA library.

Related Items

See the VI_ATTR_RSRC_MANF_ID description in this chapter. Also see the *VISA Resource Template* description in Appendix C, *Resources*.

VI_ATTR_RSRC_NAME

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViRsrc	N/A	N/A

Description

VI_ATTR_RSRC_NAME is the unique identifier for a resource compliant with the address structure shown in the following table. Optional string segments are shown in square brackets.

Interface	Syntax
VXI	VXI[*board*]::*VXI logical address*[::INSTR]
GPIB-VXI	GPIB-VXI[*board*]::*VXI logical address*[::INSTR]
GPIB	GPIB[*board*]::*primary address*[::*secondary address*][::INSTR]
ASRL	ASRL[*board*][::INSTR]
VXI	VXI[*board*]::MEMACC
GPIB-VXI	GPIB-VXI[*board*]::MEMACC

The following table shows examples of address strings as defined in the previous table.

Address String	Description
VXI0::1::INSTR	A VXI device at logical address 1 in VXI interface VXI0.
GPIB-VXI::9::INSTR	A VXI device at logical address 9 in a GPIB-VXI controlled system.
GPIB::1::0::INSTR	A GPIB device at primary address 1 and secondary address 0 in GPIB interface 0.
ASRL1::INSTR	A serial device located on port 1.

Address String	Description
VXI::MEMACC	Board-level register access to the VXI interface.
GPIB-VXI1::MEMACC	Board-level register access to GPIB-VXI interface number 1.

Related Items

See the viOpen() description in Chapter 5, *Operations*. Also see the *VISA Resource Template* description in Appendix C, *Resources*.

VI_ATTR_RSRC_SPEC_VERSION

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	`ViVersion`	0h to FFFFFFFFh	00200000h

Description

`VI_ATTR_RSRC_SPEC_VERSION` is the resource version that uniquely identifies the version of the VISA specification to which the implementation is compliant. The format of the value has the upper 12 bits as the major number of the version, the next lower 12 bits as the minor number of the version, and the lowest 8 bits as the sub-minor number of the version. The current VISA specification defines the value to be 00200000h.

Related Items

See the `VI_ATTR_RSRC_IMPL_VERSION` description in this chapter. Also see the *VISA Resource Template* description in Appendix C, *Resources*.

VI_ATTR_SEND_END_EN

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViBoolean	VI_TRUE (1) VI_FALSE (0)	VI_TRUE

Description

VI_ATTR_SEND_END_EN specifies whether to assert END during the transfer of the last byte of the buffer.

Related Items

See the *INSTR Resource* description in Appendix C, *Resources*.

VI_ATTR_SIGP_STATUS_ID

❑ Serial	❑ GPIB	❑ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only	ViUInt16	0h to FFFFh	N/A

Description

VI_ATTR_SIGP_STATUS_ID is the 16-bit Status/ID value retrieved during the IACK cycle or from the Signal register.

Related Items

See the VI_EVENT_VXI_SIGP event description in Chapter 4, *Events*. Also see the *INSTR Resource* description in Appendix C, *Resources*.

VI_ATTR_SLOT

❑ Serial	❑ GPIB	■ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViInt16	0 to 12 VI_UNKNOWN_SLOT (–1)	N/A

Description

VI_ATTR_SLOT specifies the physical slot location of the VXIbus device. If the slot number is not known, VI_UNKNOWN_SLOT is returned.

Related Items

See the *INSTR Resource* description in Appendix C, *Resources*.

VI_ATTR_SRC_ACCESS_PRIV

❑ Serial	❑ GPIB	■ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt16	VI_DATA_PRIV (0) VI_DATA_NPRIV (1) VI_PROG_PRIV (2) VI_PROG_NPRIV (3) VI_BLCK_PRIV (4) VI_BLCK_NPRIV (5) VI_D64_PRIV (6) VI_D64_NPRIV (7)	VI_DATA_PRIV

Description

VI_ATTR_SRC_ACCESS_PRIV specifies the address modifier to be used in high-level access operations, such as viIn*XX*( ) and viMoveIn*XX*( ), when reading from the source.

Related Items

See the VI_ATTR_DEST_ACCESS_PRIV, VI_ATTR_SRC_BYTE_ORDER, VI_ATTR_SRC_INCREMENT, and VI_ATTR_WIN_ACCESS_PRIV descriptions in this chapter. Also see the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix C, *Resources*.

VI_ATTR_SRC_BYTE_ORDER

❑ Serial	❑ GPIB	■ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt16	VI_BIG_ENDIAN (0) VI_LITTLE_ENDIAN (1)	VI_BIG_ENDIAN

Description

VI_ATTR_SRC_BYTE_ORDER specifies the byte order to be used in high-level access operations, such as viIn*XX*( ) and viMoveIn*XX*( ), when reading from the source.

Related Items

See the VI_ATTR_DEST_BYTE_ORDER, VI_ATTR_SRC_ACCESS_PRIV, VI_ATTR_SRC_INCREMENT, and VI_ATTR_WIN_BYTE_ORDER descriptions in this chapter. Also see the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix C, *Resources*.

VI_ATTR_SRC_INCREMENT

❑ Serial	❑ GPIB	■ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	`ViInt32`	0 to 1	1

Description

`VI_ATTR_SRC_INCREMENT` is used in the `viMoveIn`*`XX`*`()` operations to specify by how many elements the source offset is to be incremented after every transfer. The default value of this attribute is 1 (that is, the source address will be incremented by 1 after each transfer), and the `viMoveOut`*`XX`*`()` operations move from consecutive elements. If this attribute is set to 0, the `viMoveIn`*`XX`*`()` operations will always read from the same element, essentially treating the source as a FIFO register.

Related Items

See the `VI_ATTR_DEST_INCREMENT`, `VI_ATTR_SRC_ACCESS_PRIV`, and `VI_ATTR_SRC_BYTE_ORDER` descriptions in this chapter. Also see the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix C, *Resources*.

VI_ATTR_STATUS

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only	ViStatus	N/A	N/A

Description

VI_ATTR_STATUS contains the return code of the operation generating this event.

Related Items

See the VI_ATTR_BUFFER, VI_ATTR_JOB_ID, and VI_ATTR_RET_COUNT descriptions in this chapter. See the VI_EVENT_EXCEPTION and VI_EVENT_IO_COMPLETION event descriptions in Chapter 4, *Events*. Also see the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix C, *Resources*.

VI_ATTR_SUPPRESS_END_EN

■ Serial	❑ GPIB	■ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViBoolean	VI_TRUE (1) VI_FALSE (0)	VI_FALSE

Description

VI_ATTR_SUPPRESS_END_EN specifies whether to suppress the END bit termination. If this attribute is set to VI_TRUE, the END bit does not terminate read operations. If this attribute is set to VI_FALSE, the END bit terminates read operations.

Related Items

See the *INSTR Resource* description in Appendix C, *Resources*.

VI_ATTR_TERMCHAR

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt8	0 to FFh	0Ah (linefeed)

Description

VI_ATTR_TERMCHAR is the termination character. When the termination character is read and VI_ATTR_TERMCHAR_EN is enabled during a read operation, the read operation terminates.

Related Items

See the VI_ATTR_TERMCHAR_EN description in this chapter. Also see the *INSTR Resource* description in Appendix C, *Resources*.

VI_ATTR_TERMCHAR_EN

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViBoolean	VI_TRUE (1) VI_FALSE (0)	VI_FALSE

Description

VI_ATTR_TERMCHAR_EN is a flag that determines whether the read operation should terminate when a termination character is received.

Related Items

See the VI_ATTR_TERMCHAR description in this chapter. Also see the *INSTR Resource* description in Appendix C, *Resources*.

VI_ATTR_TMO_VALUE

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt32	VI_TMO_IMMEDIATE (0); 1 to FFFFFFFEh; VI_TMO_INFINITE (FFFFFFFFh)	2000

Description

VI_ATTR_TMO_VALUE specifies the minimum timeout value to use (in milliseconds) when accessing the device associated with the given session. A timeout value of VI_TMO_IMMEDIATE means that operations should never wait for the device to respond. A timeout value of VI_TMO_INFINITE disables the timeout mechanism.

Notice that the actual timeout value used by the driver may be higher than the requested one. The actual timeout value is returned when this attribute is retrieved via viGetAttribute().

Related Items

See the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix C, *Resources*.

VI_ATTR_TRIG_ID

❑ Serial	■ GPIB	■ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViInt16	**GPIB**, **Serial**: VI_TRIG_SW (–1)	VI_TRIG_SW
		VXI, **GPIB-VXI**: VI_TRIG_SW (–1); VI_TRIG_TTL0 (0) to VI_TRIG_TTL7 (7); VI_TRIG_ECL0 (8) to VI_TRIG_ECL1 (9)	VI_TRIG_SW

Description

VI_ATTR_TRIG_ID is the identifier for the current triggering mechanism.

VI_ATTR_TRIG_ID is Read/Write when the corresponding session is not enabled to receive trigger events. When the session is enabled to receive trigger events, the attribute VI_ATTR_TRIG_ID is Read Only.

Related Items

See the VI_ATTR_RECV_TRIG_ID description in this chapter. Also see the *INSTR Resource* description in Appendix C, *Resources*.

VI_ATTR_USER_DATA

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViAddr	0h to FFFFFFFFh	N/A

Description

VI_ATTR_USER_DATA is the data used privately by the application for a particular session. This data is not used by VISA for any purposes. It is provided to the application for its own use.

Related Items

See the *VISA Resource Template* description in Appendix C, *Resources*.

VI_ATTR_VXI_LA

❑ Serial	❑ GPIB	■ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViInt16	0 to 511	N/A

Description

For an INSTR session, VI_ATTR_VXI_LA specifies the logical address of the VXI or VME device used by the given session. For a MEMACC session, this attribute specifies the logical address of the local controller.

Related Items

See the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix C, *Resources*.

VI_ATTR_WIN_ACCESS

❑ Serial	❑ GPIB	■ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Local	ViUInt16	VI_NMAPPED (1) VI_USE_OPERS (2) VI_DEREF_ADDR (3)	VI_NMAPPED

Description

VI_ATTR_WIN_ACCESS specifies the modes in which the current window may be accessed.

- If VI_NMAPPED, the window is not currently mapped.
- If VI_USE_OPERS, the window is accessible through the viPeek*XX*( ) and viPoke*XX*( ) operations only.
- If VI_DEREF_ADDR, you can either use operations or directly dereference the mapped address as a pointer.

Related Items

See the VI_ATTR_WIN_ACCESS_PRIV, VI_ATTR_WIN_BASE_ADDR, VI_ATTR_WIN_BYTE_ORDER, and VI_ATTR_WIN_SIZE descriptions in this chapter. See the viMapAddress( ) description in Chapter 5, *Operations*. Also see the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix C, *Resources*.

VI_ATTR_WIN_ACCESS_PRIV

❑ Serial	❑ GPIB	■ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt16	VI_DATA_PRIV (0) VI_DATA_NPRIV (1) VI_PROG_PRIV (2) VI_PROG_NPRIV (3) VI_BLCK_PRIV (4) VI_BLCK_NPRIV (5)	VI_DATA_PRIV

Description

VI_ATTR_WIN_ACCESS_PRIV specifies the address modifier to be used in low-level access operations, such as viMapAddress(), viPeek*XX*(), and viPoke*XX*(), when accessing the mapped window.

This attribute is read/write when the corresponding session is not mapped (that is, when VI_ATTR_WIN_ACCESS is VI_NMAPPED). When the session is mapped, this attribute is read only.

Related Items

See the VI_ATTR_DEST_ACCESS_PRIV, VI_ATTR_SRC_ACCESS_PRIV, VI_ATTR_WIN_ACCESS, VI_ATTR_WIN_BASE_ADDR, VI_ATTR_WIN_BYTE_ORDER, and VI_ATTR_WIN_SIZE descriptions in this chapter. Also see the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix C, *Resources*.

VI_ATTR_WIN_BASE_ADDR

❑ Serial	❑ GPIB	■ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Local	ViBusAddress	0h to FFFFFFFFh	N/A

Description

VI_ATTR_WIN_BASE_ADDR specifies the base address of the interface bus to which this window is mapped. If the value of VI_ATTR_WIN_ACCESS is VI_NMAPPED, the value of this attribute is meaningless.

Related Items

See the VI_ATTR_WIN_ACCESS, VI_ATTR_WIN_ACCESS_PRIV, VI_ATTR_WIN_BYTE_ORDER, and VI_ATTR_WIN_SIZE descriptions in this chapter. Also see the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix C, *Resources*.

VI_ATTR_WIN_BYTE_ORDER

❑ Serial	❑ GPIB	■ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt16	VI_BIG_ENDIAN (0) VI_LITTLE_ENDIAN (1)	VI_BIG_ENDIAN

Description

VI_ATTR_WIN_BYTE_ORDER specifies the byte order to be used in low-level access operations, such as viMapAddress(), viPeek*XX*(), and viPoke*XX*(), when accessing the mapped window.

This attribute is read/write when the corresponding session is not mapped (that is, when VI_ATTR_WIN_ACCESS is VI_NMAPPED). When the session is mapped, this attribute is read only.

Related Items

See the VI_ATTR_DEST_BYTE_ORDER, VI_ATTR_SRC_BYTE_ORDER, VI_ATTR_WIN_ACCESS, VI_ATTR_WIN_ACCESS_PRIV, VI_ATTR_WIN_BASE_ADDR, and VI_ATTR_WIN_SIZE descriptions in this chapter. Also see the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix C, *Resources*.

VI_ATTR_WIN_SIZE

❑ Serial	❑ GPIB	■ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Local	ViBusSize	0h to FFFFFFFFh	N/A

Description

VI_ATTR_WIN_SIZE specifies the size of the region mapped to this window. If the value of VI_ATTR_WIN_ACCESS is VI_NMAPPED, the value of this attribute is meaningless.

Related Items

See the VI_ATTR_WIN_ACCESS, VI_ATTR_WIN_ACCESS_PRIV, VI_ATTR_WIN_BASE_ADDR, and VI_ATTR_WIN_BYTE_ORDER descriptions in this chapter. Also see the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix C, *Resources*.

VI_ATTR_WR_BUF_OPER_MODE

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt16	VI_FLUSH_ON_ACCESS (1) VI_FLUSH_WHEN_FULL (2)	VI_FLUSH_WHEN_FULL

Description

VI_ATTR_WR_BUF_OPER_MODE specifies the operational mode of the formatted I/O write buffer. When the operational mode is set to VI_FLUSH_WHEN_FULL (default), the buffer is flushed when an END indicator is written to the buffer, or when the buffer fills up. If the operational mode is set to VI_FLUSH_ON_ACCESS, the write buffer is flushed under the same conditions, and also every time a viPrintf() (or related) operation completes.

Related Items

See the VI_ATTR_RD_BUF_OPER_MODE description in this chapter. See the viPrintf() description in Chapter 5, *Operations*. Also see the *INSTR Resource* description in Appendix C, *Resources*.

4

Events

This chapter describes the VISA events. The event descriptions are listed in alphabetical order for easy reference.

Each event description contains a checkbox table below the title indicating the supported interface(s), whether Serial, GPIB, GPIB-VXI, and/or VXI; the checkbox is filled in for those that are applicable. The event description contains a brief description of the event attributes. Chapter 3, *Attributes*, contains more detailed descriptions of the event attributes.

VI_EVENT_EXCEPTION

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

Description

This event notifies the application that an error condition has occurred during an operation invocation. In VISA, exceptions are defined as events. The exception-handling model follows the event-handling model for callbacks, and is like any other event in VISA, except that the queueing and suspended handler mechanisms are not allowed.

A VISA operation generating an exception blocks until the exception handler execution is completed. However, an exception handler sometimes may prefer to terminate the program prematurely without returning the control to the operation generating the exception. VISA does not preclude an application from using a platform-specific or language-specific exception handling mechanism from within the VISA exception handler. For example, the C++ try/catch block can be used in an application in conjunction with the C++ throw mechanism from within the VISA exception handler.

When using the C++ try/catch/throw or other exception-handling mechanisms, the control will not return to the VISA system. This has some important repercussions:

- If multiple handlers were installed on the exception event, the handlers that were not invoked prior to the current handler will not be invoked for the current exception.
- The exception context will not be deleted by the VISA system when a C++ exception is used. In this case, the application should delete the exception context as soon as the application has no more use for the context, before terminating the session. An application should use the `viClose()` operation to delete the exception context.

One situation in which an exception event will not be generated is in the case of asynchronous operations. If the error is detected after the operation is posted—once the asynchronous portion has begun—the status is returned normally via the I/O completion event. However, if an error occurs before the asynchronous portion begins—the error is returned from the asynchronous operation itself—then the exception event will still be raised. This deviation is due to the fact that asynchronous operations already raise an event when they complete, and this I/O completion event may occur in the context of a separate thread previously unknown to the application. In summary, a single application event handler can easily handle error conditions arising from both exception events and failed asynchronous operations.

Event Attributes

Symbolic Name	Description
VI_ATTR_EVENT_TYPE	Unique logical identifier of the event. This attribute always has the value of VI_EVENT_EXCEPTION for this event type.
VI_ATTR_STATUS	Contains the status code returned by the operation generating the error.
VI_ATTR_OPER_NAME	Contains the name of the operation generating the event.

Related Items

See the VI_ATTR_EVENT_TYPE, VI_ATTR_STATUS, and VI_ATTR_OPER_NAME descriptions in Chapter 3, *Attributes*. See the viEnableEvent() description in Chapter 5, *Operations*. Also see the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix C, *Resources*.

VI_EVENT_IO_COMPLETION

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

Description

This event notifies the application that an asynchronous operation has completed.

Event Attributes

Symbolic Name	Description
VI_ATTR_EVENT_TYPE	Unique logical identifier of the event. This attribute always has the value of VI_EVENT_IO_COMPLETION for this event type.
VI_ATTR_STATUS	Contains the return code of the asynchronous I/O operation that has completed.
VI_ATTR_JOB_ID	Contains the job ID of the asynchronous operation that has completed.
VI_ATTR_BUFFER	Contains the address of the buffer that was used in the asynchronous operation.
VI_ATTR_RET_COUNT	Contains the actual number of elements that were asynchronously transferred.
VI_ATTR_OPER_NAME	Contains the name of the operation generating the event.

Related Items

See the VI_ATTR_EVENT_TYPE, VI_ATTR_STATUS, VI_ATTR_JOB_ID, VI_ATTR_BUFFER, VI_ATTR_RET_COUNT, and VI_ATTR_OPER_NAME descriptions in Chapter 3, *Attributes*. Also see the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix C, *Resources*.

VI_EVENT_SERVICE_REQ

❑ Serial	■ GPIB	■ GPIB-VXI	■ VXI

Description

This event notifies the application that a service request was received from the device associated with the given session.

☞ **Note** ***When you receive a*** `VI_EVENT_SERVICE_REQ`***, you must call*** `viReadSTB()` ***to guarantee delivery of future service request events on the given session.***

Event Attributes

Symbolic Name	Description
`VI_ATTR_EVENT_TYPE`	Unique logical identifier of the event. This attribute always has the value of `VI_EVENT_SERVICE_REQ` for this event type.

Related Items

See the `VI_ATTR_EVENT_TYPE` description in Chapter 3, *Attributes*. See the `viReadSTB()` description in Chapter 5, *Operations*. Also see the *INSTR Resource* description in Appendix C, *Resources*.

VI_EVENT_TRIG

❑ Serial	❑ GPIB	❑ GPIB-VXI	■ VXI

Description

This event notifies the application that a trigger interrupt was received from the device. The only triggers that can be sensed are VXI hardware triggers on the assertion edge (SYNC and ON trigger protocols only).

Event Attributes

Symbolic Name	Description
VI_ATTR_EVENT_TYPE	Unique logical identifier of the event. This attribute always has the value of VI_EVENT_TRIG for this event type.
VI_ATTR_RECV_TRIG_ID	The identifier of the triggering mechanism on which the specified trigger event was received.

Related Items

See the VI_ATTR_EVENT_TYPE and VI_ATTR_RECV_TRIG_ID descriptions in Chapter 3, *Attributes*. Also see the *INSTR Resource* description in Appendix C, *Resources*.

VI_EVENT_VXI_SIGP

❑ Serial	❑ GPIB	❑ GPIB-VXI	■ VXI

Description

This event notifies the application that a VXIbus signal or VXIbus interrupt was received from the device associated with the given session.

Event Attributes

Symbolic Name	Description
VI_ATTR_EVENT_TYPE	Unique logical identifier of the event. This attribute always has the value of VI_EVENT_VXI_SIGP for this event type.
VI_ATTR_SIGP_STATUS_ID	The 16-bit Status/ID value retrieved during the IACK cycle or from the Signal register.

Related Items

See the VI_ATTR_EVENT_TYPE and VI_ATTR_SIGP_STATUS_ID descriptions in Chapter 3, *Attributes*. Also see the *INSTR Resource* description in Appendix C, *Resources*.

VI_EVENT_VXI_VME_INTR

❑ Serial	❑ GPIB	❑ GPIB-VXI	■ VXI

Description

This event notifies the application that a VXIbus interrupt was received from the device associated with the given session.

Event Attributes

Symbolic Name	Description
VI_ATTR_EVENT_TYPE	Unique logical identifier of the event. This attribute always has the value of VI_EVENT_VXI_VME_INTR for this event type.
VI_ATTR_INTR_STATUS_ID	The 32-bit Status/ID value retrieved during the IACK cycle.
VI_ATTR_RECV_INTR_LEVEL	The VXI interrupt level on which the interrupt was received.

Related Items

See the VI_ATTR_EVENT_TYPE, VI_ATTR_INTR_STATUS_ID, and VI_ATTR_RECV_INTR_LEVEL descriptions in Chapter 3, *Attributes*. Also see the *INSTR Resource* description in Appendix C, *Resources*.

5

Operations

This chapter describes the VISA operations. The operation descriptions are listed in alphabetical order for easy reference.

Each operation description contains a checkbox table below the title indicating the supported interface(s), whether Serial, GPIB, GPIB-VXI, and/or VXI; the checkbox is filled in for those that are applicable. You will then see the operation defined in both ANSI C and Visual Basic version 4 syntax, with the parameters set in **boldface** type. A brief *Purpose* statement is followed by a table that describes each parameter and indicates whether it is an input or output parameter (or both, in some cases). The *Return Values* section describes the completion and error codes, followed by a detailed *Description* section. The *Related Items* section directs you toward related operations, events, or resource descriptions. If you want to know specifically about attributes, events, and operations of the INSTR Resource, for example, you should turn to the *INSTR Resource* section in Appendix C, *Resources*.

viAssertTrigger

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

C Syntax

```
ViStatus viAssertTrigger(ViSession vi, ViUInt16 protocol)
```

Visual Basic Syntax

```
viAssertTrigger&(ByVal vi&, ByVal protocol%)
```

Purpose

Asserts software or hardware trigger.

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
protocol	IN	Trigger protocol to use during assertion. Valid values are: VI_TRIG_PROT_DEFAULT (0), VI_TRIG_PROT_ON (1), VI_TRIG_PROT_OFF (2), and VI_TRIG_PROT_SYNC (5).

Return Values

Completion Codes	Description
VI_SUCCESS	The specified trigger was successfully asserted to the device.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given **vi** does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_INV_PROT	The protocol specified is invalid.

Error Codes	Description
VI_ERROR_TMO	Timeout expired before operation completed.
VI_ERROR_RAW_WR_PROT_VIOL	Violation of raw write protocol occurred during transfer.
VI_ERROR_RAW_RD_PROT_VIOL	Violation of raw read protocol occurred during transfer.
VI_ERROR_INP_PROT_VIOL	Device reported an input protocol error during transfer.
VI_ERROR_BERR	Bus error occurred during transfer.
VI_ERROR_LINE_IN_USE	The specified trigger line is currently in use.
VI_ERROR_NCIC	The interface associated with the given **vi** is not currently the controller in charge.
VI_ERROR_NLISTENERS	No-listeners condition is detected (both NRFD and NDAC are unasserted).
VI_ERROR_INV_SETUP	Unable to start operation because setup is invalid (due to attributes being set to an inconsistent state).

Description

The viAssertTrigger() operation will source a software or hardware trigger dependent on the interface type. For a GPIB device, the device is addressed to listen, and then the GPIB *GET* command is sent. For a VXI device, if VI_ATTR_TRIG_ID is VI_TRIG_SW, then the device is sent the Word Serial *Trigger* command; for any other values of the attribute, a hardware trigger is sent on the line that corresponds to the value of that attribute. For a serial device, if VI_ATTR_IO_PROT is VI_ASRL488, the device is sent the string "*TRG\n"; this operation is not valid for a serial device if VI_ATTR_IO_PROT is VI_NORMAL.

For GPIB, serial, and VXI software triggers, VI_TRIG_PROT_DEFAULT is the only valid protocol. For VXI hardware triggers, VI_TRIG_PROT_DEFAULT is equivalent to VI_TRIG_PROT_SYNC.

Related Items

See the VI_ATTR_TRIG_ID description in Chapter 3, *Attributes*. Also see the *INSTR Resource* description in Appendix C, *Resources*.

viBufRead

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

C Syntax

```
ViStatus viBufRead(ViSession vi, ViPBuf buf, ViUInt32 count,
                   ViPUInt32 retCount)
```

Visual Basic Syntax

```
viBufRead&(ByVal vi&, ByVal buf$, ByVal count&, retCount&)
```

Purpose

Reads data from device through the use of a formatted I/O read buffer.

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
buf	OUT	Location of a buffer to receive data from device.
count	IN	Number of bytes to be read.
retCount	OUT	Number of bytes actually transferred.

Return Values

Completion Codes	Description
VI_SUCCESS	The operation completed successfully and the END indicator was received (for interfaces that have END indicators). This completion code is returned regardless of whether the termination character is received or the number of bytes read is equal to **count**.

Completion Codes	Description
VI_SUCCESS_TERM_CHAR	The specified termination character was read but no END indicator was received. This completion code is returned regardless of whether the number of bytes read is equal to **count**.
VI_SUCCESS_MAX_CNT	The number of bytes read is equal to **count**. No END indicator was received and no termination character was read.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given **vi** does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_TMO	Timeout expired before operation completed.
VI_ERROR_IO	An unknown I/O error occurred during transfer.

Description

The `viBufRead()` operation is similar to `viRead()` and does not perform any kind of data formatting. It differs from `viRead()` in that the data is read from the formatted I/O read buffer—the same buffer used by `viScanf()` and related operations—rather than directly from the device. You can intermix this operation with `viScanf()`, but you should not mix it with `viRead()`.

`VI_NULL` is a special value for the **retCount** parameter. If you pass `VI_NULL` for **retCount**, the number of bytes transferred is not returned. You may find this useful if you need to know only whether the operation succeeded or failed.

Related Items

See the `viRead()` and `viBufWrite()` descriptions in this chapter. Also see the *INSTR Resource* description in Appendix C, *Resources*.

viBufWrite

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

C Syntax

```
ViStatus viBufWrite(ViSession vi, ViBuf buf, ViUInt32 count,
                    ViPUInt32 retCount)
```

Visual Basic Syntax

```
viBufWrite&(ByVal vi&, ByVal buf$, ByVal count&, retCount&)
```

Purpose

Writes data to a formatted I/O write buffer synchronously.

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
buf	IN	Location of a block of data.
count	IN	Number of bytes to be written.
retCount	OUT	Number of bytes actually transferred.

Return Values

Completion Codes	Description
VI_SUCCESS	Operation completed successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given **vi** does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_TMO	Timeout expired before operation completed.

Error Codes	Description
VI_ERROR_INV_SETUP	Unable to start write operation because setup is invalid (due to attributes being set to an inconsistent state).
VI_ERROR_IO	An unknown I/O error occurred during transfer.

Description

The viBufWrite() operation is similar to viWrite() and does not perform any kind of data formatting. It differs from viWrite() in that the data is written to the formatted I/O write buffer—the same buffer used by viPrintf() and related operations—rather than directly to the device. You can intermix this operation with viPrintf(), but you should not mix it with viWrite().

If this operation returns VI_ERROR_TMO, the write buffer for the specified session is cleared.

VI_NULL is a special value for the **retCount** parameter. If you pass VI_NULL for **retCount**, the number of bytes transferred is not returned. You may find this useful if you need to know only whether the operation succeeded or failed.

Related Items

See the viWrite() and viBufRead() descriptions in this chapter. Also see the *INSTR Resource* description in Appendix C, *Resources*.

viClear

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

C Syntax

```
ViStatus viClear(ViSession vi)
```

Visual Basic Syntax

```
viClear&(ByVal vi&)
```

Purpose

Clears a device.

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.

Return Values

Completion Codes	Description
VI_SUCCESS	Operation completed successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given **vi** does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_TMO	Timeout expired before operation completed.
VI_ERROR_RAW_WR_PROT_VIOL	Violation of raw write protocol occurred during transfer.
VI_ERROR_RAW_RD_PROT_VIOL	Violation of raw read protocol occurred during transfer.

Error Codes	Description
VI_ERROR_BERR	Bus error occurred during transfer.
VI_ERROR_NCIC	The interface associated with the given **vi** is not currently the controller in charge.
VI_ERROR_NLISTENERS	No-listeners condition is detected (both NRFD and NDAC are unasserted).
VI_ERROR_INV_SETUP	Unable to start operation because setup is invalid (due to attributes being set to an inconsistent state).

Description

The viClear() operation performs an IEEE 488.1-style clear of the device (for VXI, the Word Serial Clear command is used; for GPIB systems, the Selected Device Clear command is used). For a serial device, if VI_ATTR_IO_PROT is VI_ASRL488, the device is sent the string "*CLS\n"; this operation is not valid for a serial device if VI_ATTR_IO_PROT is VI_NORMAL. Invoking viClear() on an INSTR Resource will also discard the read and write buffers used by the formatted I/O services for that session.

Related Items

See the *INSTR Resource* description in Appendix C, *Resources*.

viClose

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

C Syntax

```
ViStatus viClose(ViObject vi)
```

Visual Basic Syntax

```
viClose&(ByVal vi&)
```

Purpose

Closes the specified session, event, or find list.

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session, event, or find list.

Return Values

Completion Codes	Description
VI_SUCCESS	Session closed successfully.
VI_WARN_NULL_OBJECT	The specified object reference is uninitialized.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given object reference is invalid.
VI_ERROR_CLOSING_FAILED	Unable to deallocate the previously allocated data structures corresponding to this session or object reference.

Description

The `viClose()` operation closes a session, event, or a find list. In this process all the data structures that had been allocated for the specified **vi** are freed.

Related Items

See the `viOpen()`, `viOpenDefaultRM()`, `viFindRsrc()`, and `viWaitOnEvent()` descriptions in this chapter. Also see the *VISA Resource Template* description in Appendix C, *Resources*.

viDisableEvent

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

C Syntax

```
ViStatus viDisableEvent(ViSession vi, ViEventType eventType,
                        ViUInt16 mechanism)
```

Visual Basic Syntax

```
viDisableEvent&(ByVal vi&, ByVal eventType&, ByVal mechanism%)
```

Purpose

Disables notification of the specified event type(s) via the specified mechanism(s).

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
eventType	IN	Logical event identifier.
mechanism	IN	Specifies event handling mechanisms to be disabled. The queuing mechanism is disabled by specifying `VI_QUEUE` (1), and the callback mechanism is disabled by specifying `VI_HNDLR` (2) or `VI_SUSPEND_HNDLR` (4). It is possible to disable both mechanisms simultaneously by specifying `VI_ALL_MECH` (FFFFh).

Return Values

Completion Codes	Description
`VI_SUCCESS`	Event disabled successfully.
`VI_SUCCESS_EVENT_DIS`	Specified event is already disabled for at least one of the specified mechanisms.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_INV_EVENT	Specified event type is not supported by the resource.
VI_ERROR_INV_MECH	Invalid mechanism specified.

Description

The viDisableEvent() operation disables servicing of an event identified by the **eventType** parameter for the mechanisms specified in the **mechanism** parameter. This operation prevents *new* event occurrences from being added to the queue(s). However, event occurrences already existing in the queue(s) are not flushed. Use viDiscardEvents() if you want to discard events remaining in the queue(s).

Specifying VI_ALL_ENABLED_EVENTS for the **eventType** parameter allows a session to stop receiving all events. The session can stop receiving queued events by specifying VI_QUEUE. Applications can stop receiving callback events by specifying either VI_HNDLR or VI_SUSPEND_HNDLR. Specifying VI_ALL_MECH disables both the queuing and callback mechanisms.

Related Items

See the viEnableEvent() description in this chapter. Also see the *VISA Resource Template* description in Appendix C, *Resources*.

viDiscardEvents

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

C Syntax

```
ViStatus viDiscardEvents(ViSession vi, ViEventType eventType,
                         ViUInt16 mechanism)
```

Visual Basic Syntax

```
viDiscardEvents&(ByVal vi&, ByVal eventType&, ByVal mechanism%)
```

Purpose

Discards event occurrences for specified event types and mechanisms in a session.

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
eventType	IN	Logical event identifier.
mechanism	IN	Specifies the mechanisms for which the events are to be discarded. The `VI_QUEUE` (1) value is specified for the queuing mechanism and the `VI_SUSPEND_HNDLR` (4) value is specified for the pending events in the callback mechanism. It is possible to specify both mechanisms simultaneously by specifying `VI_ALL_MECH` (FFFFh).

Return Values

Completion Codes	Description
`VI_SUCCESS`	Event queue flushed successfully.
`VI_SUCCESS_QUEUE_EMPTY`	Operation completed successfully, but queue was already empty.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_INV_EVENT	Specified event type is not supported by the resource.
VI_ERROR_INV_MECH	Invalid mechanism specified.

Description

The viDiscardEvents() operation discards all pending occurrences of the specified event types and mechanisms from the specified session. Specifying VI_ALL_ENABLED_EVENTS for the **eventType** parameter discards events of every type that is enabled for the given session. The information about all the event occurrences which have not yet been handled is discarded. This operation is useful to remove event occurrences that an application no longer needs. The discarded event occurrences are not available to a session at a later time. This operation does not apply to event contexts that have already been delivered to the application.

Related Items

See the viEnableEvent(), viDisableEvent(), and viWaitOnEvent() descriptions in this chapter. Also see the *VISA Resource Template* description in Appendix C, *Resources*.

viEnableEvent

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

C Syntax

```
ViStatus viEnableEvent(ViSession vi, ViEventType eventType,
                       ViUInt16 mechanism, ViEventFilter context)
```

Visual Basic Syntax

```
viEnableEvent&(ByVal vi&, ByVal eventType&, ByVal mechanism%,
               ByVal context&)
```

Purpose

Enables notification of a specified event.

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
eventType	IN	Logical event identifier.
mechanism	IN	Specifies event handling mechanisms to be enabled. The queuing mechanism is enabled by specifying `VI_QUEUE` (1), and the callback mechanism is enabled by specifying `VI_HNDLR` (2) or `VI_SUSPEND_HNDLR` (4). It is possible to enable both mechanisms simultaneously by specifying *bit-wise OR* of `VI_QUEUE` and one of the two mode values for the callback mechanism.
context	IN	`VI_NULL` (0).

Return Values

Completion Codes	Description
`VI_SUCCESS`	Event enabled successfully.
`VI_SUCCESS_EVENT_EN`	Specified event is already enabled for at least one of the specified mechanisms.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_INV_EVENT	Specified event type is not supported by the resource.
VI_ERROR_INV_MECH	Invalid mechanism specified for the event.
VI_ERROR_INV_CONTEXT	Specified event context is invalid.
VI_ERROR_HNDLR_NINSTALLED	A handler is not currently installed for the specified event. The session cannot be enabled for the VI_HNDLR mode of the callback mechanism.

Description

The viEnableEvent() operation enables notification of an event identified by the **eventType** parameter for mechanisms specified in the **mechanism** parameter. The specified session can be enabled to queue events by specifying VI_QUEUE. Applications can enable the session to invoke a callback function to execute the handler by specifying VI_HNDLR. The applications are required to install at least one handler to be enabled for this mode. Specifying VI_SUSPEND_HNDLR enables the session to receive callbacks, but the invocation of the handler is deferred to a later time. Successive calls to this operation replace the old callback mechanism with the new callback mechanism. Specifying VI_ALL_ENABLED_EVENTS for the **eventType** parameter refers to all events which have previously been enabled on this session, making it easier to switch between the two callback mechanisms for multiple events.

Related Items

See the viDisableEvent() and viWaitOnEvent() descriptions in this chapter. Also see the viInstallHandler() and viUninstallHandler() descriptions in this chapter for information about installing and uninstalling event handlers. See Chapter 4, *Events*, for a list of events that you can enable. Also see the *VISA Resource Template* description in Appendix C, *Resources*.

viEventHandler

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

C Syntax

```
ViStatus _VI_FUNCH viEventHandler(ViSession vi, ViEventType
                                  eventType, ViEvent context,
                                  ViAddr userHandle)
```

Visual Basic Syntax

N/A

Purpose

Event service handler procedure prototype.

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
eventType	IN	Logical event identifier.
context	IN	A handle specifying the unique occurrence of an event.
userHandle	IN	A value specified by an application that can be used for identifying handlers uniquely in a session for an event.

Return Values

Completion Codes	Description
VI_SUCCESS	Event handled successfully.
VI_SUCCESS_NCHAIN	Event handled successfully. Do not invoke any other handlers on this session for this event.

Description

viEventHandler() is not an actual VISA operation. Rather, it is the prototype for a user event handler that is installed with the viInstallHandler() operation. The user handler is called whenever a session receives an event and is enabled for handling events in the VI_HNDLR mode. The handler services the event and returns VI_SUCCESS on completion.

The VISA system automatically invokes the `viClose()` operation on the event context when a user handler returns.

Because the event context must still be valid after the user handler returns (so that VISA can free it up), an application should not invoke the `viClose()` operation on an event context passed to a user handler.

- **Note for advanced users**: If the user handler will not return to VISA, the application should call `viClose()` on the event context to manually delete the event object. This situation may occur when a handler throws a C++ exception in response to a VISA exception event.

Normally, an application should always return `VI_SUCCESS` from all callback handlers. If a specific handler does not want other handlers to be invoked for the given event for the given session, it should return `VI_SUCCESS_NCHAIN`. No return value from a handler on one session will affect callbacks on other sessions. Future versions of VISA (or specific implementations of VISA) may take actions based on other return values, so a user should return `VI_SUCCESS` from handlers unless there is a specific reason to do otherwise.

Related Items

See `viInstallHandler()` and `viUninstallHandler()` descriptions in this chapter. Also see the *VISA Resource Template* description in Appendix C, *Resources*.

viFindNext

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

C Syntax

```
ViStatus viFindNext(ViFindList findList, ViChar instrDesc[])
```

Visual Basic Syntax

```
viFindNext&(ByVal findList&, ByVal instrDesc$)
```

Purpose

Returns the next resource from the list of resources found during a previous call to `viFindRsrc()`.

Parameters

Name	Direction	Description
findList	IN	Describes a find list. This parameter must be created by `viFindRsrc()`.
instrDesc	OUT	Returns a string identifying the location of a device. Strings can then be passed to `viOpen()` to establish a session to the given device.

Return Values

Completion Codes	Description
VI_SUCCESS	Resource(s) found.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given object reference is invalid.
VI_ERROR_NSUP_OPER	The given **findList** does not support this operation.
VI_ERROR_RSRC_NFOUND	There are no more matches.

Description

The `viFindNext()` operation returns the next device found in the list created by `viFindRsrc()`. The list is referenced by the handle that was returned by `viFindRsrc()`.

☞ **Note** ***The size of the* instrDesc *parameter should be at least 256 bytes.***

Related Items

See the `viFindRsrc()` description in this chapter. Also see the *VISA Resource Manager* description in Appendix C, *Resources*.

viFindRsrc

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

C Syntax

```
ViStatus viFindRsrc(ViSession sesn, ViString expr,
                    ViPFindList findList, ViPUInt32 retcnt,
                    ViChar instrDesc[])
```

Visual Basic Syntax

```
viFindRsrc&(ByVal sesn&, ByVal expr$, findList&, retcnt&,
            ByVal instrDesc$)
```

Purpose

Queries a VISA system to locate the resources associated with a specified interface.

Parameters

Name	Direction	Description
sesn	IN	Resource Manager session (should always be the session returned from `viOpenDefaultRM()`).
expr	IN	This is a regular expression followed by an optional logical expression. Refer to the discussion of the Description String in the *Description* section of this operation.
findList	OUT	Returns a handle identifying this search session. This handle will be used as in input in `viFindNext()`.
retcnt	OUT	Number of matches.
instrDesc	OUT	Returns a string identifying the location of a device. Strings can then be passed to `viOpen()` to establish a session to the given device.

Return Values

Completion Codes	Description
`VI_SUCCESS`	Resource(s) found.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given **sesn** does not support this operation. This operation is supported only by a Resource Manager session.
VI_ERROR_INV_EXPR	Invalid expression specified for search.
VI_ERROR_RSRC_NFOUND	Specified expression does not match any devices.

Description

The viFindRsrc() operation matches the value specified in the **expr** parameter with the resources available for a particular interface. A regular expression is a string consisting of ordinary characters as well as special characters. You use a regular expression to specify patterns to match in a given string; in other words, it is a search criterion. The viFindRsrc() operation uses a case-insensitive compare feature when matching resource names against the regular expression specified in **expr**. For example, calling viFindRsrc() with "VXI?*INSTR" would return the same resources as invoking it with "vxi?*instr".

On successful completion, this function returns the first resource found in the list and returns a count (**retcnt**) to indicate if there were more resources found for the designated interface. This function also returns, in the **findList** parameter, a handle to a find list. This handle points to the list of resources and it must be used as an input to viFindNext(). When this handle is no longer needed, it should be passed to viClose(). Notice that **retcnt** and **findList** are optional parameters. This is useful if only the first match is important, and the number of matches is not needed. If you specify VI_NULL in the **findList** parameter and the operation completes successfully, VISA automatically invokes viClose() on the find list handle rather than returning it to the application.

☞ **Note** ***The size of the* instrDesc *parameter should be at least 256 bytes.***

The search criteria specified in the **expr** parameter has two parts: a regular expression over a resource string, and an optional logical expression over attribute values. The regular expression is matched against the resource strings of resources known to the VISA Resource Manager. If the resource string matches the regular expression, the attribute values of the resource are then matched against the expression over attribute values. If the match is successful, the resource has met the search criteria and gets added to the list of resources found.

Special Characters and Operators	Meaning
?	Matches any one character.
\	Makes the character that follows it an ordinary character instead of special character. For example, when a question mark follows a backslash (\?), it matches the ? character instead of any one character.
[list]	Matches any one character from the enclosed list. You can use a hyphen to match a range of characters.
[^list]	Matches any character not in the enclosed list. You can use a hyphen to match a range of characters.
*	Matches 0 or more occurrences of the preceding character or expression.
+	Matches 1 or more occurrences of the preceding character or expression.
exp|exp	Matches either the preceding or following expression. The or operator | matches the entire expression that precedes or follows it and not just the character that precedes or follows it. For example, VXI|GPIB means (VXI)|(GPIB), not VX(I|G)PIB.
(exp)	Grouping characters or expressions.

Regular Expression	Sample Matches
GPIB?*INSTR	Matches GPIB0::2::INSTR, GPIB1::1::1::INSTR, and GPIB-VXI1::8::INSTR.
GPIB[0-9]*::?*INSTR	Matches GPIB0::2::INSTR and GPIB1::1::1::INSTR but not GPIB-VXI1::8::INSTR.
GPIB[^0]::?*INSTR	Matches GPIB1::1::1::INSTR but not GPIB0::2::INSTR or GPIB12::8::INSTR.
VXI?*INSTR	Matches VXI0::1::INSTR but not GPIB-VXI0::1::INSTR.
GPIB-VXI?*INSTR	Matches GPIB-VXI0::1::INSTR but not VXI0::1::INSTR.
?*VXI[0-9]*::?*INSTR	Matches VXI0::1::INSTR and GPIB-VXI0::1::INSTR.
ASRL[0-9]*::?*INSTR	Matches ASRL1::INSTR but not VXI0::5::INSTR.
ASRL1+::INSTR	Matches ASRL1::INSTR and ASRL11::INSTR but not ASRL2::INSTR.
(GPIB|VXI)?*INSTR	Matches GPIB1::5::INSTR and VXI0::3::INSTR but not ASRL2::INSTR.
(GPIB0|VXI0)::1::INSTR	Matches GPIB0::1::INSTR and VXI0::1::INSTR.
?*INSTR	Matches all INSTR (device) resources.
?*VXI[0-9]*::?*MEMACC	Matches VXI0::MEMACC and GPIB-VXI1::MEMACC.
VXI0::?*	Matches VXI0::1::INSTR, VXI0::2::INSTR, and VXI0::MEMACC.
?*	Matches all resources.

By using the optional attribute expression, you can construct flexible and powerful expressions with the use of logical ANDs (&&), ORs(||), and NOTs (!). You can use equal (==) and unequal (!=) comparators to compare attributes of any type, and other inequality comparators (>, <, >=, <=) to compare attributes of numeric type. Use only global attributes in the attribute expression. Local attributes are not allowed in the logical expression part of the **expr** parameter.

Expr Parameter	Meaning
`GPIB[0-9]*::?*::?*::INSTR {VI_ATTR_GPIB_SECONDARY_ADDR > 0 && VI_ATTR_GPIB_SECONDARY_ADDR < 10}`	Find all GPIB devices that have secondary addresses from 1 to 9.
`ASRL?*INSTR{VI_ATTR_ASRL_BAUD == 9600}`	Find all serial ports configured at 9600 baud.
`?*VXI?INSTR{VI_ATTR_MANF_ID == 0xFF6 && !(VI_ATTR_VXI_LA ==0 || VI_ATTR_SLOT <= 0)}`	Find all VXI instrument resources having manufacturer ID FF6 and which are not logical address 0, slot 0, or external controllers.

Related Items

See the `viClose()` and `viFindNext()` descriptions in this chapter. Also see the *VISA Resource Manager* description in Appendix C, *Resources*.

viFlush

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

C Syntax

```
ViStatus viFlush(ViSession vi, ViUInt16 mask)
```

Visual Basic Syntax

```
viFlush&(ByVal vi&, ByVal mask%)
```

Purpose

Manually flushes the specified buffers associated with formatted I/O operations and/or serial communication.

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
mask	IN	Specifies the action to be taken with flushing the buffer. Refer to the *Description* section for more information.

Return Values

Completion Codes	Description
VI_SUCCESS	Buffers flushed successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_IO	Could not perform read/write operation because of I/O error.

Error Codes	Description
VI_ERROR_TMO	The read/write operation was aborted because timeout expired while operation was in progress.
VI_ERROR_INV_MASK	The specified **mask** does not specify a valid flush operation on read/write resource.

Description

The value of **mask** can be one of the following flags:

Flag	Interpretation
VI_READ_BUF (1)	Discard the read buffer contents. If data was present in the read buffer and no END-indicator was present, read from the device until encountering an END indicator (which causes the loss of data). This action resynchronizes the next viScanf() call to read a <TERMINATED RESPONSE MESSAGE>. (Refer to the IEEE 488.2 standard.)
VI_READ_BUF_DISCARD (4)	Discard the read buffer contents (does not perform any I/O to the device).
VI_WRITE_BUF (2)	Flush the write buffer by writing all buffered data to the device.
VI_WRITE_BUF_DISCARD (8)	Discard the write buffer contents (does not perform any I/O to the device).
VI_ASRL_IN_BUF (16)	Discard the receive buffer contents (same as VI_ASRL_IN_BUF_DISCARD).
VI_ASRL_IN_BUF_DISCARD (64)	Discard the receive buffer contents (does not perform any I/O to the device).
VI_ASRL_OUT_BUF (32)	Flush the transmit buffer by writing all buffered data to the device.
VI_ASRL_OUT_BUF_DISCARD (128)	Discard the transmit buffer contents (does not perform any I/O to the device).

It is possible to combine any of these read flags and write flags for different buffers by ORing the flags. However, combining two flags for the same buffer in the same call to viFlush() is illegal.

Notice that when using formatted I/O operations with a serial device, a flush of the formatted I/O buffers also causes the corresponding serial communication buffers to be flushed. For example, calling `viFlush()` with `VI_WRITE_BUF` also flushes the `VI_ASRL_OUT_BUF`.

Related Items

See the `viSetBuf()` description in this chapter. Also see the *INSTR Resource* description in Appendix C, *Resources*.

viGetAttribute

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

C Syntax

```
ViStatus viGetAttribute(ViObject vi, ViAttr attribute,
                        void * attrState)
```

Visual Basic Syntax

```
viGetAttribute&(ByVal vi&, ByVal attribute&, attrState as Any)
```

Purpose

Retrieves the state of an attribute.

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session, event, or find list.
attribute	IN	Resource attribute for which the state query is made.
attrState	OUT	The state of the queried attribute for a specified resource. The interpretation of the returned value is defined by the individual object.

Return Values

Completion Codes	Description
VI_SUCCESS	Attribute retrieved successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given object reference is invalid.
VI_ERROR_NSUP_ATTR	The specified attribute is not defined by the referenced object.

Description

The viGetAttribute() operation is used to retrieve the state of an attribute for the specified session, event, or find list.

The output parameter **attrState** is of the type of the attribute actually being retrieved. For example, when retrieving an attribute that is defined as a `ViBoolean`, your application should pass a reference to a variable of type `ViBoolean`. Similarly, if the attribute is defined as being `ViUInt32`, your application should pass a reference to a variable of type `ViUInt32`.

Related Items

See the `viSetAttribute()` description in this chapter. Also see the *VISA Resource Template* description in Appendix C, *Resources*, and the attribute descriptions in Chapter 3, *Attributes*.

viGpibControlREN

❑ Serial	■ GPIB	❑ GPIB-VXI	❑ VXI

C Syntax

```
ViStatus viGpibControlREN(ViSession vi, ViUInt16 mode)
```

Visual Basic Syntax

```
viGpibControlREN&(ByVal vi&, ByVal mode%)
```

Purpose

Controls the state of the GPIB Remote Enable (REN) interface line, and optionally the remote/local state of the device.

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
mode	IN	Specifies the state of the REN line and optionally the device remote/local state. See the *Description* section for actual values.

Return Values

Completion Codes	Description
VI_SUCCESS	Operation completed successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given **vi** does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_NCIC	The interface associated with this session is not currently the controller in charge.

Error Codes	Description
VI_ERROR_NLISTENERS	No-listeners condition is detected (both NRFD and NDAC are unasserted.
VI_ERROR_NSYS_CNTLR	The interface associated with this session is not the system controller.
VI_ERROR_INV_MODE	The value specified by the **mode** parameter is invalid.

Description

The viGpibControlREN() operation asserts or unasserts the GPIB REN interface line according to the specified mode. The mode can also specify whether the device associated with this session should be placed in locate state (before deasserting REN) or remote state (after asserting REN). This operation is valid only if the GPIB interface associated with the session specified by **vi** is currently the system controller.

The following table lists special values for the **mode** parameter.

Value	Description
VI_GPIB_REN_DEASSERT	Deassert REN line.
VI_GPIB_REN_ASSERT	Assert REN line.
VI_GPIB_REN_DEASSERT_GTL	Send the Go To Local (GTL) command and deassert REN line.
VI_GPIB_REN_ASSERT_ADDRESS	Assert REN line and address device.

Related Items

See the *INSTR Resource* description in Appendix C, *Resources*.

viIn8/viIn16/viIn32

❑ Serial	❑ GPIB	■ GPIB-VXI	■ VXI

C Syntax

```
ViStatus viIn8(ViSession vi, ViUInt16 space,
               ViBusAddress offset, ViPUInt8 val8)
ViStatus viIn16(ViSession vi, ViUInt16 space,
                ViBusAddress offset, ViPUInt16 val16)
ViStatus viIn32(ViSession vi, ViUInt16 space,
                ViBusAddress offset, ViPUInt32 val32)
```

Visual Basic Syntax

```
viIn8&(ByVal vi&, ByVal space%, ByVal offset&, val8 as Byte)
viIn16&(ByVal vi&, ByVal space%, ByVal offset&, val16%)
viIn32&(ByVal vi&, ByVal space%, ByVal offset&, val32&)
```

Purpose

Reads in an 8-bit, 16-bit, or 32-bit value from the specified memory space and offset.

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
space	IN	Specifies the address space. Refer to the table included in the *Description* section for more information.
offset	IN	Offset (in bytes) of the address or register from which to read.
val8, **val16**, or **val32**	OUT	Data read from bus (8 bits for `viIn8()`, 16 bits for `viIn16()`, and 32 bits for `viIn32()`).

Return Values

Completion Codes	Description
`VI_SUCCESS`	Operation completed successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given **vi** does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_BERR	Bus error occurred during transfer.
VI_ERROR_INV_SPACE	Invalid address space specified.
VI_ERROR_INV_OFFSET	Invalid offset specified.
VI_ERROR_NSUP_OFFSET	Specified offset is not accessible from this hardware.
VI_ERROR_NSUP_WIDTH	Specified width is not supported by this hardware.
VI_ERROR_NSUP_ALIGN_OFFSET	The specified offset is not properly aligned for the access width of the operation.
VI_ERROR_INV_SETUP	Unable to start operation because setup is invalid (due to attributes being set to an inconsistent state).

Description

The viIn*XX*() operations use the specified address space to read in 8, 16, or 32 bits of data, respectively, from the specified offset. These operations do not require viMapAddress() to be called prior to their invocation.

The following table lists the valid entries for specifying address space.

Value	Description
VI_A16_SPACE (1)	Address the A16 address space of the VXI/MXI bus.
VI_A24_SPACE (2)	Address the A24 address space of the VXI/MXI bus.
VI_A32_SPACE (3)	Address the A32 address space of the VXI/MXI bus.

INSTR Specific

Notice that **offset** specified in the viIn8(), viIn16(), and viIn32() operations for an INSTR resource is the offset address relative to the device's allocated address base for the corresponding address space that was specified. For example, if **space** specifies VI_A16_SPACE, then **offset** specifies the offset from the logical address base address of

the specified VXI device. If **space** specifies `VI_A24_SPACE` or `VI_A32_SPACE`, then **offset** specifies the offset from the base address of the VXI device's memory space allocated by the VXI Resource Manager within VXI A24 or A32 space.

MEMACC Specific

For a MEMACC resource, the **offset** parameter specifies an absolute address.

Related Items

See the `viOut8/viOut16/viOut32()` descriptions in this chapter. Also see the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix C, *Resources*.

viInstallHandler

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

C Syntax

```
ViStatus viInstallHandler(ViSession vi, ViEventType eventType,
                          ViHndlr handler, ViAddr userHandle)
```

Visual Basic Syntax

N/A

Purpose

Installs handlers for event callbacks.

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
eventType	IN	Logical event identifier.
handler	IN	Interpreted as a valid reference to a handler to be installed by a client application.
userHandle	IN	A value specified by an application that can be used for identifying handlers uniquely for an event type.

Return Values

Completion Codes	Description
VI_SUCCESS	Event handler installed successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_INV_EVENT	Specified event type is not supported by the resource.

Error Codes	Description
VI_ERROR_INV_HNDLR_REF	The given handler reference is invalid.
VI_ERROR_HNDLR_NINSTALLED	The handler was not installed. This may be returned if an application attempts to install multiple handlers for the same event on the same session.

Description

The `viInstallHandler()` operation allows applications to install handlers on sessions. The handler specified in the **handler** parameter is installed along with any previously installed handlers for the specified event. Applications can specify a value in the **userHandle** parameter that is passed to the handler on its invocation. VISA identifies handlers uniquely using the handler reference and this value.

VISA allows applications to install multiple handlers for an event type on the same session. You can install multiple handlers through multiple invocations of the `viInstallHandler()` operation, where each invocation adds to the previous list of handlers. If more than one handler is installed for an event type, each of the handlers is invoked on every occurrence of the specified event(s). VISA specifies that the handlers are invoked in Last In First Out (LIFO) order.

Related Items

See the `viEventHandler()`, `viEnableEvent()`, and `viUninstallHandler()` descriptions. Also see the *VISA Resource Template* description in Appendix C, *Resources*.

viLock

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

C Syntax

```
ViStatus viLock(ViSession vi, ViAccessMode lockType, ViUInt32
                timeout, ViKeyId requestedKey, ViChar accesskey[])
```

Visual Basic Syntax

```
viLock&(ByVal vi&, ByVal lockType&, ByVal timeout&, ByVal
        requestedKey$, ByVal accesskey$)
```

Purpose

Establishes an access mode to the specified resource.

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
lockType	IN	Specifies the type of lock requested, either VI_EXCLUSIVE_LOCK (1) or VI_SHARED_LOCK (2).
timeout	IN	Absolute time period (in milliseconds) that a resource waits to get unlocked by the locking session before returning an error.
requestedKey	IN	This parameter is not used and should be set to VI_NULL when **lockType** is VI_EXCLUSIVE_LOCK. When **lockType** is VI_SHARED_LOCK, a session can either set this parameter to VI_NULL so that VISA generates an **accessKey**, or the session can suggest an **accessKey** to use for the shared lock. Refer to the *Description* section for more details.
accessKey	OUT	This parameter should be set to VI_NULL when **lockType** is VI_EXCLUSIVE_LOCK. When **lockType** is VI_SHARED_LOCK, the resource returns a unique access key for the lock if the operation succeeds. This **accessKey** can then be passed to other sessions to share the lock.

Return Values

Completion Codes	Description
VI_SUCCESS	Specified access mode was acquired.
VI_SUCCESS_NESTED_EXCLUSIVE	Specified access mode is successfully acquired, and this session has nested exclusive locks.
VI_SUCCESS_NESTED_SHARED	Specified access mode is successfully acquired, and this session has nested shared locks.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_RSRC_LOCKED	Specified type of lock cannot be obtained because the resource is already locked with a lock type incompatible with the lock requested.
VI_ERROR_INV_LOCK_TYPE	The specified type of lock is not supported by this resource.
VI_ERROR_INV_ACCESS_KEY	The **requestedKey** value passed in is not a valid access key to the specified resource.
VI_ERROR_TMO	Specified type of lock could not be obtained within the specified timeout period.

Description

This operation is used to obtain a lock on the specified resource. The caller can specify the type of lock requested—exclusive or shared lock—and the length of time the operation will suspend while waiting to acquire the lock before timing out. This operation can also be used for sharing and nesting locks.

The **requestedKey** and the **accessKey** parameters apply only to shared locks. These parameters are not applicable when using the lock type VI_EXCLUSIVE_LOCK; in this case, **requestedKey** and **accessKey** should be set to VI_NULL. VISA allows user applications to specify a key to be used for lock sharing, through the use of the **requestedKey** parameter. Alternatively, a user application can pass VI_NULL for the **requestedKey** parameter when obtaining a shared lock, in which case VISA will generate a unique access key and return it through the **accessKey** parameter. If a user application does specify a **requestedKey** value, VISA will try to use this value for the **accessKey**. As long as the resource is not locked, VISA will use the **requestedKey** as the access key and grant the lock. When the operation succeeds, the **requestedKey** will be copied into the user buffer referred to by the **accessKey** parameter.

Note ***If requesting a*** `VI_SHARED_LOCK`***, the size of the*** **accessKey** ***parameter should be at least 256 bytes.***

The session that gained a shared lock can pass the **accessKey** to other sessions for the purpose of sharing the lock. The session wanting to join the group of sessions sharing the lock can use the key as an input value to the **requestedKey** parameter. VISA will add the session to the list of sessions sharing the lock, as long as the **requestedKey** value matches the **accessKey** value for the particular resource. The session obtaining a shared lock in this manner will then have the same access privileges as the original session that obtained the lock.

It is also possible to obtain nested locks through this operation. To acquire nested locks, invoke the `viLock()` operation with the same lock type as the previous invocation of this operation. For each session, `viLock()` and `viUnlock()` share a lock count, which is initialized to 0. Each invocation of `viLock()` for the same session (and for the same **lockType**) increases the lock count. In the case of a shared lock, it returns with the same **accessKey** every time. When a session locks the resource a multiple number of times, it is necessary to invoke the `viUnlock()` operation an equal number of times in order to unlock the resource. That is, the lock count increments for each invocation of `viLock()`, and decrements for each invocation of `viUnlock()`. A resource is actually unlocked only when the lock count is 0.

Related Items

See the `viUnlock()` description in this chapter. Also see the *VISA Resource Template* description in Appendix C, *Resources*.

viMapAddress

❑ Serial	❑ GPIB	■ GPIB-VXI	■ VXI

C Syntax

```
ViStatus viMapAddress(ViSession vi, ViUInt16 mapSpace,
                      ViBusAddress mapBase, ViBusSize mapSize,
                      ViBoolean access, ViAddr suggested,
                      ViPAddr address)
```

Visual Basic Syntax

```
viMapAddress&(ByVal vi&, ByVal mapSpace%, ByVal mapBase&, ByVal
              mapSize&, ByVal access%, ByVal suggested&, address&)
```

Purpose

Maps the specified memory space into the process's address space.

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
mapSpace	IN	Specifies the address space to map. Refer to the *Description* section for more information.
mapBase	IN	Offset (in bytes) of the memory to be mapped. Refer to the *Description* section for more information.
mapSize	IN	Amount of memory to map (in bytes).
access	IN	`VI_FALSE` (0).
suggested	IN	If **suggested** parameter is not `VI_NULL` (0), the operating system attempts to map the memory to the address specified in **suggested**. There is no guarantee, however, that the memory will be mapped to that address. This operation may map the memory into an address region different from **suggested**.
address	OUT	Address in your process space where the memory was mapped.

Return Values

Completion Codes	Description
VI_SUCCESS	Mapping successful.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given **vi** does not support this operation.
VI_ERROR_INV_SPACE	Invalid address space specified.
VI_ERROR_INV_OFFSET	Invalid offset specified.
VI_ERROR_NSUP_OFFSET	Specified region is not accessible from this hardware.
VI_ERROR_TMO	viMapAddress() could not acquire resource or perform mapping before the timer expired.
VI_ERROR_INV_SIZE	Invalid size of window specified.
VI_ERROR_ALLOC	Unable to allocate window of at least the requested size.
VI_ERROR_INV_ACC_MODE	Invalid access mode.
VI_ERROR_WINDOW_MAPPED	The specified session already contains a mapped window.
VI_ERROR_INV_SETUP	Unable to start operation because setup is invalid (due to attributes being set to an inconsistent state).

Description

The viMapAddress() operation maps in a specified memory space. The memory space that is mapped is dependent on the type of interface specified by the **vi** parameter and the **mapSpace** parameter. The **address** parameter returns the address in your process space where memory is mapped. The following table lists the valid entries for the **mapSpace** parameter.

Value	Description
VI_A16_SPACE (1)	Maps the A16 address space of the VXI/MXI bus.

Value	Description
VI_A24_SPACE (2)	Maps the A24 address space of the VXI/MXI bus.
VI_A32_SPACE (3)	Maps the A32 address space of the VXI/MXI bus.

INSTR Specific

Notice that **mapBase** specified in the `viMapAddress()` operation for an INSTR resource is the offset address relative to the device's allocated address base for the corresponding address space that was specified. For example, if **mapSpace** specifies `VI_A16_SPACE`, then **mapBase** specifies the offset from the logical address base address of the specified VXI device. If **mapSpace** specifies `VI_A24_SPACE` or `VI_A32_SPACE`, then **mapBase** specifies the offset from the base address of the VXI device's memory space allocated by the VXI Resource Manager within VXI A24 or A32 space.

MEMACC Specific

For a MEMACC resource, the **mapBase** parameter specifies an absolute address.

Related Items

See the `viUnmapAddress()` description in this chapter. Also see the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix C, *Resources*.

viMemAlloc

❑ Serial	❑ GPIB	■ GPIB-VXI	■ VXI

C Syntax

```
ViStatus viMemAlloc(ViSession vi, ViBusSize size, ViPBusAddress
                    offset)
```

Visual Basic Syntax

```
viMemAlloc&(ByVal vi&, ByVal size&, offset&)
```

Purpose

Allocates memory from a device's memory region.

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
size	IN	Specifies the size of the allocation.
offset	OUT	Returns the offset of the allocated device memory.

Return Values

Completion Codes	Description
VI_SUCCESS	Operation completed successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given **vi** does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_INV_SIZE	Invalid size specified.

Error Codes	Description
VI_ERROR_ALLOC	Unable to allocate shared memory block of the requested size.
VI_ERROR_MEM_NSHARED	The device does not export any memory.

Description

The viMemAlloc() operation returns an offset into a device's memory region that has been allocated for use by this session. If the device to which the given **vi** refers is located on the local interface card, the memory can be allocated either on the device itself or on the computer's system memory.

The memory region referenced by the **offset** that is returned from this operation can be accessed with the high-level operations viMoveIn*XX*() and viMoveOut*XX*(), or it can be mapped using viMapAddress().

Related Items

See the viMapAddress(), viMemFree(), viMoveIn8/viMoveIn16/viMoveIn32(), and viMoveOut8/viMoveOut16/viMoveOut32() descriptions in this chapter. Also see the *INSTR Resource* description in Appendix C, *Resources*.

viMemFree

❑ Serial	❑ GPIB	■ GPIB-VXI	■ VXI

C Syntax

```
ViStatus viMemFree(ViSession vi, ViBusAddress offset)
```

Visual Basic Syntax

```
viMemFree&(ByVal vi&, ByVal offset&)
```

Purpose

Frees memory previously allocated using the `viMemAlloc()` operation.

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
offset	IN	Specifies the memory previously allocated with `viMemAlloc()`.

Return Values

Completion Codes	Description
VI_SUCCESS	Operation completed successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given **vi** does not support this operation.
VI_ERROR_INV_OFFSET	Invalid **offset** specified.
VI_ERROR_WINDOW_MAPPED	The specified **offset** is currently in use by `viMapAddress()`.

Description

The `viMemFree()` operation frees the memory previously allocated using `viMemAlloc()`. If the specified **offset** has been mapped using `viMapAddress()`, it must be unmapped before it can be freed.

Related Items

See the `viMapAddress()`, `viMemAlloc()`, and `viUnmapAddress()` descriptions in this chapter. Also see the *INSTR Resource* description in Appendix C, *Resources*.

viMove

❑ Serial	❑ GPIB	■ GPIB-VXI	■ VXI

C Syntax

```
ViStatus viMove(ViSession vi, ViUInt16 srcSpace, ViBusAddress
                srcOffset, ViUInt16 srcWidth, ViUInt16 destSpace,
                ViBusAddress destOffset, ViUInt16 destWidth,
                ViBusSize length)
```

Visual Basic Syntax

```
viMove&(ByVal vi&, ByVal srcSpace%, ByVal srcOffset&, ByVal
        srcWidth%, ByVal destSpace%, ByVal destOffset&, ByVal
        destWidth%, ByVal length&)
```

Purpose

Moves a block of data.

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
srcSpace	IN	Specifies the address space of the source.
srcOffset	IN	Offset of the starting address or register from which to read.
srcWidth	IN	Specifies the data width of the source.
destSpace	IN	Specifies the address space of the destination.
destOffset	IN	Offset of the starting address or register to which to write.
destWidth	IN	Specifies the data width of the destination.
length	IN	Number of elements to transfer, where the data width of the elements to transfer is identical to the source data width.

Return Values

Completion Codes	Description
VI_SUCCESS	Operation completed successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given **vi** does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_BERR	Bus error occurred during transfer.
VI_ERROR_INV_SPACE	Invalid source or destination address space specified.
VI_ERROR_INV_OFFSET	Invalid source or destination offset specified.
VI_ERROR_INV_WIDTH	Invalid source or destination width specified.
VI_ERROR_NSUP_OFFSET	Specified source or destination offset is not accessible from this hardware.
VI_ERROR_NSUP_VAR_WIDTH	Cannot support source and destination widths that are different.
VI_ERROR_INV_SETUP	Unable to start operation because setup is invalid (due to attributes being set to an inconsistent state).
VI_ERROR_NSUP_WIDTH	Specified width is not supported by this hardware.
VI_ERROR_NSUP_ALIGN_OFFSET	The specified offset is not properly aligned for the access width of the operation.
VI_ERROR_INV_LENGTH	Invalid length specified.

Description

The `viMove()` operation moves data from the specified source to the specified destination. The source and the destination can either be local memory or the offset of the interface with which this MEMACC resource is associated. This operation uses the specified data width and address space. In some systems, such as VXI, users can specify additional settings for the transfer, such as byte order and access privilege, by manipulating the appropriate attributes.

The following table lists the valid entries for specifying address space.

Value	Description
VI_A16_SPACE (1)	Address the A16 address space of the VXI/MXI bus.
VI_A24_SPACE (2)	Address the A24 address space of the VXI/MXI bus.
VI_A32_SPACE (3)	Address the A32 address space of the VXI/MXI bus.
VI_LOCAL_SPACE (0)	Address process-local memory (using a virtual address).

The following table lists the valid entries for specifying widths.

Value	Description
VI_WIDTH_8 (1)	Performs 8-bit (D08) transfers.
VI_WIDTH_16 (2)	Performs 16-bit (D16) transfers.
VI_WIDTH_32 (4)	Performs 32-bit (D32) transfers.

INSTR Specific

If **srcSpace** is not VI_LOCAL_SPACE, then **srcOffset** is a relative address of the device associated with the given INSTR resource. Similarly, if **destSpace** is not VI_LOCAL_SPACE, then **destOffset** is a relative address of the device associated with the given INSTR resource.

The primary intended use of this operation with an INSTR session is to synchronously move data to or from the device. Therefore, either the **srcSpace** or **destSpace** parameter will usually be VI_LOCAL_SPACE.

MEMACC Specific

The **destOffset** and **srcOffset** parameters specify absolute addresses. Notice also that the **length** specified in the viMove() operation is the number of elements (of the size corresponding to the **srcWidth** parameter) to transfer, beginning at the specified offsets. Therefore, **srcOffset + length*srcWidth** cannot exceed the total amount of memory exported by the given **srcSpace**. Similarly, **destOffset** + **length*srcWidth** cannot exceed the total amount of memory exported by the given **destSpace**.

Related Items

See the viMoveAsync() description in this chapter. See the VI_ATTR_DEST_INCREMENT and VI_ATTR_SRC_INCREMENT descriptions in Chapter 3, *Attributes*. Also see the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix C, *Resources*.

viMoveAsync

❑ Serial	❑ GPIB	■ GPIB-VXI	■ VXI

C Syntax

```
ViStatus viMoveAsync(ViSession vi, ViUInt16 srcSpace, ViBusAddress
                     srcOffset, ViUInt16 srcWidth, ViUInt16
                     destSpace, ViBusAddress destOffset, ViUInt16
                     destWidth, ViBusSize length, ViPJobId jobId)
```

Visual Basic Syntax

N/A

Purpose

Moves a block of data asynchronously.

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
srcSpace	IN	Specifies the address space of the source.
srcOffset	IN	Offset of the starting address or register from which to read.
srcWidth	IN	Specifies the data width of the source.
destSpace	IN	Specifies the address space of the destination.
destOffset	IN	Offset of the starting address or register to which to write.
destWidth	IN	Specifies the data width of the destination.
length	IN	Number of elements to transfer, where the data width of the elements to transfer is identical to the source data width.
jobId	OUT	Job identifier of this asynchronous move operation.

Return Values

Completion Codes	Description
VI_SUCCESS	Asynchronous operation successfully queued.
VI_SUCCESS_SYNC	Operation performed synchronously.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given **vi** does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_QUEUE_ERROR	Unable to queue move operation.

Description

The viMoveAsync() operation asynchronously moves data from the specified source to the specified destination. This operation queues up the transfer in the system, then it returns immediately without waiting for the transfer to carry out or complete. When the transfer terminates, a VI_EVENT_IO_COMPLETION event is generated, which indicates the status of the transfer.

This operation returns **jobId**, which you can use either with viTerminate() to abort the operation or with VI_EVENT_IO_COMPLETION events to identify which asynchronous move operations completed.

The source and the destination can either be local memory or the offset of the interface with which this INSTR or MEMACC resource is associated. This operation uses the specified data width and address space. In some systems, such as VXI, users can specify additional settings for the transfer, such as byte order and access privilege, by manipulating the appropriate attributes.

The following table lists the valid entries for specifying address space.

Value	Description
VI_A16_SPACE (1)	Address the A16 address space of the VXI/MXI bus.
VI_A24_SPACE (2)	Address the A24 address space of the VXI/MXI bus.

Value	Description
VI_A32_SPACE (3)	Address the A32 address space of the VXI/MXI bus.
VI_LOCAL_SPACE (0)	Address process-local memory (using a virtual address).

The following table lists the valid entries for specifying widths.

Value	Description
VI_WIDTH_8 (1)	Performs 8-bit (D08) transfers.
VI_WIDTH_16 (2)	Performs 16-bit (D16) transfers.
VI_WIDTH_32 (4)	Performs 32-bit (D32) transfers.

INSTR Specific

If **srcSpace** is not VI_LOCAL_SPACE, then **srcOffset** is a relative address of the device associated with the given INSTR resource. Similarly, if **destSpace** is not VI_LOCAL_SPACE, then **destOffset** is a relative address of the device associated with the given INSTR resource.

The primary intended use of this operation with an INSTR session is to asynchronously move data to or from the device. Therefore, either the **srcSpace** or **destSpace** parameter will usually be VI_LOCAL_SPACE.

MEMACC Specific

The **destOffset** and **srcOffset** parameters specify absolute addresses. Notice also that the **length** specified in the viMoveAsync() operation is the number of elements (of the size corresponding to the **srcWidth** parameter) to transfer, beginning at the specified offsets. Therefore, **srcOffset + length*srcWidth** cannot exceed the total amount of memory exported by the given **srcSpace**. Similarly, **destOffset** + **length*srcWidth** cannot exceed the total amount of memory exported by the given **destSpace**.

Related Items

See the viMove() description in this chapter. Also see the VI_ATTR_DEST_INCREMENT and VI_ATTR_SRC_INCREMENT descriptions in Chapter 3, *Attributes*. See the VI_EVENT_IO_COMPLETION description in Chapter 4, *Events*. Also see the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix C, *Resources*.

viMoveIn8/viMoveIn16/viMoveIn32

❑ Serial	❑ GPIB	■ GPIB-VXI	■ VXI

C Syntax

```
ViStatus viMoveIn8(ViSession vi, ViUInt16 space, ViBusAddress
                   offset, ViBusSize length, ViAUInt8 buf8)
ViStatus viMoveIn16(ViSession vi, ViUInt16 space, ViBusAddress
                    offset, ViBusSize length, ViAUInt16 buf16)
ViStatus viMoveIn32(ViSession vi, ViUInt16 space, ViBusAddress
                    offset, ViBusSize length, ViAUInt32 buf32)
```

Visual Basic Syntax

```
viMoveIn8&(ByVal vi&, ByVal space%, ByVal offset&, ByVal length&,
           buf8 as Byte)
viMoveIn16&(ByVal vi&, ByVal space%, ByVal offset&, ByVal length&,
            buf16%)
viMoveIn32&(ByVal vi&, ByVal space%, ByVal offset&, ByVal length&,
            buf32&)
```

Purpose

Moves a block of data from the specified address space and offset to local memory.

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
space	IN	Specifies the address space. Refer to the table included in the *Description* section.
offset	IN	Offset (in bytes) of the starting address to read.
length	IN	Number of elements to transfer, where the data width of the elements to transfer is identical to data width (8, 16, or 32 bits).
buf8, **buf16**, or **buf32**	OUT	Data read from bus (8 bits for `viMoveIn8()`, 16 bits for `viMoveIn16()`, and 32 bits for `viMoveIn32()`).

Return Values

Completion Codes	Description
VI_SUCCESS	Operation completed successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given **vi** does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_BERR	Bus error occurred during transfer.
VI_ERROR_INV_SPACE	Invalid address space specified.
VI_ERROR_INV_OFFSET	Invalid offset specified.
VI_ERROR_NSUP_OFFSET	Specified offset is not accessible from this hardware.
VI_ERROR_NSUP_WIDTH	Specified width is not supported by this hardware.
VI_ERROR_INV_LENGTH	Invalid length specified.
VI_ERROR_NSUP_ALIGN_OFFSET	The specified offset is not properly aligned for the access width of the operation.
VI_ERROR_INV_SETUP	Unable to start operation because setup is invalid (due to attributes being set to an inconsistent state).

Description

The viMoveIn*XX*() operations use the specified address space to read in 8, 16, or 32 bits of data, respectively, from the specified offset. These operations do not require viMapAddress() to be called prior to their invocation.

The following table lists the valid entries for specifying address space.

Value	Description
VI_A16_SPACE (1)	Address the A16 address space of the VXI/MXI bus.

Value	Description
VI_A24_SPACE (2)	Address the A24 address space of the VXI/MXI bus.
VI_A32_SPACE (3)	Address the A32 address space of the VXI/MXI bus.

INSTR Specific

Notice that **offset** specified in the viMoveIn8(), viMoveIn16(), and viMoveIn32() operations for an INSTR resource is the offset address relative to the device's allocated address base for the corresponding address space that was specified. For example, if **space** specifies VI_A16_SPACE, then **offset** specifies the offset from the logical address base address of the specified VXI device. If **space** specifies VI_A24_SPACE or VI_A32_SPACE, then **offset** specifies the offset from the base address of the VXI device's memory space allocated by the VXI Resource Manager within VXI A24 or A32 space.

Notice also that the **length** specified in the viMoveIn*XX*() operations for an INSTR resource is the number of elements (of the **size** corresponding to the operation) to transfer, beginning at the specified **offset**. Therefore, **offset + length*size** cannot exceed the amount of memory exported by the device in the given **space**.

MEMACC Specific

For a MEMACC resource, the **offset** parameter specifies an absolute address.

Notice also that the **length** specified in the viMoveIn*XX*() operations for a MEMACC resource is the number of elements (of the **size** corresponding to the operation) to transfer, beginning at the specified **offset**. Therefore, **offset + length*size** cannot exceed the total amount of memory available in the given **space**.

Related Items

See the viMoveOut8/viMoveOut16/viMoveOut32() descriptions in this chapter. See the VI_ATTR_SRC_INCREMENT description in Chapter 3, *Attributes*. Also see the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix C, *Resources*.

viMoveOut8/viMoveOut16/viMoveOut32

❑ Serial	❑ GPIB	■ GPIB-VXI	■ VXI

C Syntax

```
ViStatus viMoveOut8(ViSession vi, ViUInt16 space, ViBusAddress
                    offset, ViBusSize length, ViAUInt8 buf8)
ViStatus viMoveOut16(ViSession vi, ViUInt16 space, ViBusAddress
                     offset, ViBusSize length, ViAUInt16 buf16)
ViStatus viMoveOut32(ViSession vi, ViUInt16 space, ViBusAddress
                     offset, ViBusSize length, ViAUInt32 buf32)
```

Visual Basic Syntax

```
viMoveOut8&(ByVal vi&, ByVal space%, ByVal offset&, ByVal
            length&,buf8 as Byte)
viMoveOut16&(ByVal vi&, ByVal space%, ByVal offset&, ByVal
             length&, buf16%)
viMoveOut32&(ByVal vi&, ByVal space%, ByVal offset&, ByVal
             length&, buf32&)
```

Purpose

Moves a block of data from local memory to the specified address space and offset.

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
space	IN	Specifies the address space. Refer to the table included in the *Description* section.
offset	IN	Offset (in bytes) of the device to write to.
length	IN	Number of elements to transfer, where the data width of the elements to transfer is identical to data width (8, 16, or 32 bits).
buf8, **buf16**, or **buf32**	IN	Data to write bus (8 bits for `viMoveOut8()`, 16 bits for `viMoveOut16()`, and 32 bits for `viMoveOut32()`).

Return Values

Completion Codes	Description
VI_SUCCESS	Operation completed successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given **vi** does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_BERR	Bus error occurred during transfer.
VI_ERROR_INV_SPACE	Invalid address space specified.
VI_ERROR_INV_OFFSET	Invalid offset specified.
VI_ERROR_NSUP_OFFSET	Specified offset is not accessible from this hardware.
VI_ERROR_NSUP_WIDTH	Specified width is not supported by this hardware.
VI_ERROR_INV_LENGTH	Invalid length specified.
VI_ERROR_NSUP_ALIGN_OFFSET	The specified offset is not properly aligned for the access width of the operation.
VI_ERROR_INV_SETUP	Unable to start operation because setup is invalid (due to attributes being set to an inconsistent state).

Description

The viMoveOut*XX*() operations use the specified address space to write 8, 16, or 32 bits of data, respectively, to the specified offset. These operations do not require viMapAddress() to be called prior to their invocation.

The following table lists the valid entries for specifying address space.

Value	Description
VI_A16_SPACE (1)	Address the A16 address space of the VXI/MXI bus.

Value	Description
VI_A24_SPACE (2)	Address the A24 address space of the VXI/MXI bus.
VI_A32_SPACE (3)	Address the A32 address space of the VXI/MXI bus.

INSTR Specific

Notice that **offset** specified in the `viMoveOut8()`, `viMoveOut16()`, and `viMoveOut32()` operations for an INSTR resource is the offset address relative to the device's allocated address base for the corresponding address space that was specified. For example, if **space** specifies `VI_A16_SPACE`, then **offset** specifies the offset from the logical address base address of the specified VXI device. If **space** specifies `VI_A24_SPACE` or `VI_A32_SPACE`, then **offset** specifies the offset from the base address of the VXI device's memory space allocated by the VXI Resource Manager within VXI A24 or A32 space.

Notice also that the **length** specified in the `viMoveIn`*XX*`()` operations for an INSTR resource is the number of elements (of the **size** corresponding to the operation) to transfer, beginning at the specified **offset**. Therefore, **offset + length*size** cannot exceed the amount of memory exported by the device in the given **space**.

MEMACC Specific

For a MEMACC resource, the **offset** parameter specifies an absolute address.

Notice also that the **length** specified in the `viMoveOut`*XX*`()` operations for a MEMACC resource is the number of elements (of the **size** corresponding to the operation) to transfer, beginning at the specified **offset**. Therefore, **offset + length*size** cannot exceed the total amount of memory available in the given **space**.

Related Items

See the `viMoveIn8/viMoveIn16/viMoveIn32()` descriptions in this chapter. See the `VI_ATTR_SRC_INCREMENT` description in Chapter 3, *Attributes*. Also see the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix C, *Resources*.

viOpen

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

C Syntax

```
ViStatus viOpen(ViSession sesn, ViRsrc rsrcName, ViAccessMode
                accessMode, ViUInt32 timeout, ViPSession vi)
```

Visual Basic Syntax

```
viOpen&(ByVal sesn&, ByVal rsrcName$, ByVal accessMode&,
        ByVal timeout&, vi&)
```

Purpose

Opens a session to the specified resource.

Parameters

Name	Direction	Description
sesn	IN	Resource Manager session (should always be a session returned from `viOpenDefaultRM()`).
rsrcName	IN	Unique symbolic name of a resource. See the *Description* section for more information.
accessMode	IN	Specifies the mode by which the resource is to be accessed. See the *Description* section for valid values. If the parameter value is `VI_NULL`, the session uses VISA-supplied default values. Multiple access modes can be used simultaneously by specifying a *bit-wise OR* of the values other than `VI_NULL`.
timeout	IN	Specifies the maximum time period (in milliseconds) that this operation waits before returning an error.
vi	OUT	Unique logical identifier reference to a session.

Return Values

Completion Codes	Description
VI_SUCCESS	Session opened successfully.
VI_SUCCESS_DEV_NPRESENT	Session opened successfully, but the device at the specified address is not responding.
VI_WARN_CONFIG_NLOADED	The specified configuration either does not exist or could not be loaded; using VISA-specified defaults.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given **sesn** does not support this operation. This operation is supported only by a Resource Manager session.
VI_ERROR_INV_RSRC_NAME	Invalid resource reference specified. Parsing error.
VI_ERROR_INV_ACC_MODE	Invalid access mode.
VI_ERROR_RSRC_NFOUND	Insufficient location information or resource not present in the system.
VI_ERROR_ALLOC	Insufficient system resources to open a session.
VI_ERROR_RSRC_BUSY	The resource is valid, but VISA cannot currently access it.
VI_ERROR_RSRC_LOCKED	Specified type of lock cannot be obtained because the resource is already locked with a lock type incompatible with the lock requested.
VI_ERROR_TMO	A session to the resource could not be obtained within the specified timeout period.
VI_ERROR_LIBRARY_NFOUND	A code library required by VISA could not be located or loaded.

Description

The `viOpen()` operation opens a session to the specified resource. It returns a session identifier that can be used to call any other operations of that resource. The address string passed to `viOpen()` must uniquely identify a resource. The following table shows the grammar for the address string. Optional string segments are shown in square brackets ([ ]).

Interface	Syntax
VXI	VXI[*board*]::*VXI logical address*[::INSTR]
GPIB-VXI	GPIB-VXI[*board*]::*VXI logical address*[::INSTR]
GPIB	GPIB[*board*]::*primary address*[::*secondary address*][::INSTR]
ASRL	ASRL[*board*][::INSTR]
VXI	VXI[*board*]::MEMACC
GPIB-VXI	GPIB-VXI[*board*]::MEMACC

The VXI keyword is used for VXI instruments via either embedded or MXIbus controllers. The GPIB-VXI keyword is used for a GPIB-VXI controller. The GPIB keyword can be used to establish communication with a GPIB device. The ASRL keyword is used to establish communication with an asynchronous serial (such as RS-232) device.

The following table shows the default value for optional string segments.

Optional String Segments	Default Value
board	0
secondary address	none

The following table shows examples of address strings.

Address String	Description
VXI0::1::INSTR	A VXI device at logical address 1 in VXI interface VXI0.
GPIB-VXI::9::INSTR	A VXI device at logical address 9 in a GPIB-VXI controlled system.
GPIB::1::0::INSTR	A GPIB device at primary address 1 and secondary address 0 in GPIB interface 0.
ASRL1::INSTR	A serial device attached to interface ASRL1.
VXI::MEMACC	Board-level register access to the VXI interface.
GPIB-VXI1::MEMACC	Board-level register access to GPIB-VXI interface number 1.

The value VI_EXCLUSIVE_LOCK (1) is used to acquire an exclusive lock immediately upon opening a session; if a lock cannot be acquired, the session is closed and an error is returned. The value VI_LOAD_CONFIG (4) is used to configure attributes to values specified by some external configuration utility.

Related Items

See the viClose() and viOpenDefaultRM() descriptions in this chapter. Also see the *VISA Resource Manager* description in Appendix C, *Resources*.

viOpenDefaultRM

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

C Syntax

```
ViStatus viOpenDefaultRM(ViPSession sesn)
```

Visual Basic Syntax

```
viOpenDefaultRM&(sesn&)
```

Purpose

This function returns a session to the Default Resource Manager resource.

Parameters

Name	Direction	Description
sesn	OUT	Unique logical identifier to a Default Resource Manager session.

Return Values

Completion Codes	Description
VI_SUCCESS	Session to the Default Resource Manager resource created successfully.

Error Codes	Description
VI_ERROR_SYSTEM_ERROR	The VISA system failed to initialize.
VI_ERROR_ALLOC	Insufficient system resources to create a session to the Default Resource Manager resource.
VI_ERROR_INV_SETUP	Some implementation-specific configuration file is corrupt or does not exist.
VI_ERROR_LIBRARY_NFOUND	A code library required by VISA could not be located or loaded.

Description

The `viOpenDefaultRM()` function must be called before any VISA operations can be invoked. The first call to this function initializes the VISA system, including the Default Resource Manager resource, and also returns a session to that resource. Subsequent calls to this function return unique sessions to the same Default Resource Manager resource.

When a Resource Manager session is passed to `viClose()`, not only is that session closed, but also all find lists and device sessions (which that Resource Manager session was used to create) are closed.

Related Items

See the `viOpen()`, `viClose()`, and `viFindRsrc()` descriptions in this chapter. Also see the *VISA Resource Manager* description in Appendix C, *Resources*.

viOut8/viOut16/viOut32

❑ Serial	❑ GPIB	■ GPIB-VXI	■ VXI

C Syntax

```
ViStatus viOut8(ViSession vi, ViUInt16 space,
                ViBusAddress offset, ViUInt8 val8)
ViStatus viOut16(ViSession vi, ViUInt16 space,
                 ViBusAddress offset, ViUInt16 val16)
ViStatus viOut32(ViSession vi, ViUInt16 space,
                 ViBusAddress offset, ViUInt32 val32)
```

Visual Basic Syntax

```
viOut8&(ByVal vi&, ByVal space%, ByVal offset&, ByVal val8 as Byte)
viOut16&(ByVal vi&, ByVal space%, ByVal offset&, ByVal val16%)
viOut32&(ByVal vi&, ByVal space%, ByVal offset&, ByVal val32&)
```

Purpose

Writes an 8-bit, 16-bit, or 32-bit value to the specified memory space and offset.

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
space	IN	Specifies the address space. Refer to the table included in the *Description* section for more information.
offset	IN	Offset (in bytes) of the address or register to which to read.
val8, **val16**, or **val32**	IN	Data to write to bus (8 bits for `viOut8()`, 16 bits for `viOut16()`, and 32 bits for `viOut32()`).

Return Values

Completion Codes	Description
`VI_SUCCESS`	Operation completed successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given **vi** does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_BERR	Bus error occurred during transfer.
VI_ERROR_INV_SPACE	Invalid address space specified.
VI_ERROR_INV_OFFSET	Invalid offset specified.
VI_ERROR_NSUP_OFFSET	Specified offset is not accessible from this hardware.
VI_ERROR_NSUP_WIDTH	Specified width is not supported by this hardware.
VI_ERROR_NSUP_ALIGN_OFFSET	The specified offset is not properly aligned for the access width of the operation.
VI_ERROR_INV_SETUP	Unable to start operation because setup is invalid (due to attributes being set to an inconsistent state).

Description

The viOut*XX*() operations use the specified address space to write 8, 16, or 32 bits of data, respectively, to the specified offset. These operations do not require viMapAddress() to be called prior to their invocation.

The following table lists the valid entries for specifying address space.

Value	Description
VI_A16_SPACE (1)	Address the A16 address space of the VXI/MXI bus.
VI_A24_SPACE (2)	Address the A24 address space of the VXI/MXI bus.
VI_A32_SPACE (3)	Address the A32 address space of the VXI/MXI bus.

INSTR Specific

Notice that **offset** specified in the viOut8(), viOut16(), and viOut32() operations for an INSTR resource is the offset address relative to the device's allocated address base for the corresponding address space that was specified. For example, if **space** specifies VI_A16_SPACE, then **offset** specifies the offset from the logical address base address of

the specified VXI device. If **space** specifies `VI_A24_SPACE` or `VI_A32_SPACE`, then **offset** specifies the offset from the base address of the VXI device's memory space allocated by the VXI Resource Manager within VXI A24 or A32 space.

MEMACC Specific

For a MEMACC resource, the **offset** parameter specifies an absolute address.

Related Items

See the `viIn8/viIn16/viIn32()` descriptions in this chapter. Also see the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix C, *Resources*.

viPeek8/viPeek16/viPeek32

❑ Serial	❑ GPIB	■ GPIB-VXI	■ VXI

C Syntax

```
void viPeek8(ViSession vi, ViAddr addr, ViPUInt8 val8)
void viPeek16(ViSession vi, ViAddr addr, ViPUInt16 val16)
void viPeek32(ViSession vi, ViAddr addr, ViPUInt32 val32)
```

Visual Basic Syntax

```
viPeek8(ByVal vi&, ByVal addr&, val8 as Byte)
viPeek16(ByVal vi&, ByVal addr&, val16%)
viPeek32(ByVal vi&, ByVal addr&, val32&)
```

Purpose

Reads an 8-bit, 16-bit, or 32-bit value from the specified address.

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
addr	IN	Source address to read the value.
val8, **val16**, or **val32**	OUT	Data read from bus (8 bits for `viPeek8()`, 16 bits for `viPeek16()`, and 32 bits for `viPeek32()`).

Return Values

None

Description

The `viPeek`*XX*`()` operations read an 8-bit, 16-bit, or 32-bit value, respectively, from the address location specified in **addr**. The address must be a valid memory address in the current process mapped by a previous `viMapAddress()` call.

Related Items

See the `viMapAddress()` and `viPoke8/viPoke16/viPoke32()` descriptions. See the `VI_ATTR_WIN_ACCESS` description in Chapter 3, *Attributes*. Also see the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix C, *Resources*.

viPoke8/viPoke16/viPoke32

❑ Serial	❑ GPIB	■ GPIB-VXI	■ VXI

C Syntax

```
void viPoke8(ViSession vi, ViAddr addr, ViUInt8 val8)
void viPoke16(ViSession vi, ViAddr addr, ViUInt16 val16)
void viPoke32(ViSession vi, ViAddr addr, ViUInt32 val32)
```

Visual Basic Syntax

```
viPoke8(ByVal vi&, ByVal addr&, ByVal val8 as Byte)
viPoke16(ByVal vi&, ByVal addr&, ByVal val16%)
viPoke32(ByVal vi&, ByVal addr&, ByVal val32&)
```

Purpose

Writes an 8-bit, 16-bit, or 32-bit value to the specified address.

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
addr	IN	Destination address to store the value.
val8, **val16**, or **val32**	IN	Value to be stored (8 bits for `viPoke8()`, 16 bits for `viPoke16()`, and 32 bits for `viPoke32()`).

Return Values

None

Description

The `viPoke`*XX*`()` operations store the content of an 8-bit, 16-bit, or 32-bit value, respectively, to the address pointed to by **addr**. The address must be a valid memory address in the current process mapped by a previous `viMapAddress()` call.

Related Items

See the `viMapAddress()` and `viPeek8/viPeek16/viPeek32()` descriptions. See the `VI_ATTR_WIN_ACCESS` description in Chapter 3, *Attributes*. Also see the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix C, *Resources*.

viPrintf

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

C Syntax

```
ViStatus viPrintf(ViSession vi, ViString writeFmt, ...)
```

Visual Basic Syntax

N/A

Purpose

Converts, formats, and sends the parameters (designated by**...**) to the device as specified by the format string.

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
writeFmt	IN	String describing the format for arguments.
…	IN	Parameters to which the format string is applied.

Return Values

Completion Codes	Description
VI_SUCCESS	Parameters were successfully formatted.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_IO	Could not perform write operation because of I/O error.
VI_ERROR_TMO	Timeout expired before write operation completed.

Error Codes	Description
VI_ERROR_INV_FMT	A format specifier in the **writeFmt** string is invalid.
VI_ERROR_NSUP_FMT	A format specifier in the **writeFmt** string is not supported.
VI_ERROR_ALLOC	The system could not allocate a formatted I/O buffer because of insufficient resources.

Description

The `viPrintf()` operation sends data to a device as specified by the format string. Before sending the data, the operation formats the arguments in the parameter list as specified in the **writeFmt** string. The `viWrite()` operation performs the actual low-level I/O to the device. As a result, you should not use the `viWrite()` and `viPrintf()` operations in the same session.

The **writeFmt** string can include regular character sequences, special formatting characters, and special format specifiers. The regular characters (including white spaces) are written to the device unchanged. The special characters consist of '\' (backslash) followed by a character. The format specifier sequence consists of '%' (percent) followed by an optional modifier (flag), followed by a format code.

Special Formatting Characters

Special formatting character sequences send special characters. The following table lists the special characters and describes what they send to the device.

Formatting Character	Character Sent to Device
\n	Sends the ASCII LF character. The END identifier will also be automatically sent.
\r	Sends an ASCII CR character.
\t	Sends an ASCII TAB character.
\###	Sends the ASCII character specified by the octal value.
\x##	Sends the ASCII character specified by the hexadecimal value.
\"	Sends the ASCII double-quote (") character.
\\	Sends a backslash (\) character.

Format Specifiers

The format specifiers convert the next parameter in the sequence according to the modifier and format code, after which the formatted data is written to the specified device. The format specifier takes the following syntax:

%[modifiers]*format code*

where *format code* specifies which data type the argument is represented in. Modifiers are optional codes that describe the target data.

In the following tables, a ‘d’ format code refers to all conversion codes of type *integer* (‘d’, ‘i’, ‘o’, ‘u’, ‘x’, ‘X’), unless specified as %d only. Similarly, an ‘f’ format code refers to all conversion codes of type *float* (‘f’, ‘e’, ‘E’, ‘g’, ‘G’), unless specified as %f only.

Every conversion command starts with the % character and ends with a conversion character (format code). Between the % character and the format code, the following modifiers can appear in the sequence.

ANSI C Standard Modifiers

Modifier	Supported with Format Code	Description
An integer specifying *field width*.	d, f, s format codes	This specifies the minimum field width of the converted argument. If an argument is shorter than the *field width*, it will be padded on the left (or on the right if the - flag is present). Special case: For the @H, @Q, and @B flags, the *field width* includes the #H, #Q, and #B strings, respectively. An asterisk (*) may be present in lieu of a field width modifier, in which case an extra **arg** is used. This **arg** must be an integer representing the *field width*.

Modifier	Supported with Format Code	Description
An integer specifying *precision*.	d, f, s format codes	The *precision* string consists of a string of decimal digits. A . (decimal point) must prefix the *precision* string. The *precision* string specifies the following: a. The minimum number of digits to appear for the @1, @H, @Q, and @B flags and the i, o, u, x, and X format codes. b. The maximum number of digits after the decimal point in case of f format codes. c. The maximum numbers of characters for the string (s) specifier. d. Maximum significant digits for g format code. An asterisk (*) may be present in lieu of a *precision* modifier, in which case an extra **arg** is used. This **arg** must be an integer representing the *precision* of a numeric field.
An argument length modifier. h, l, L, z, and Z are legal values. (z and Z are not ANSI C standard modifiers.)	h (d, b, B format codes) l (d, f, b, B format codes) L (f format code) z (b, B format codes) Z (b, B format codes)	The argument length modifiers specify one of the following: a. The h modifier promotes the argument to a short or unsigned short, depending on the format code type. b. The l modifier promotes the argument to a long or unsigned long. c. The L modifier promotes the argument to a long double parameter. d. The z modifier promotes the argument to an array of floats. e. The Z modifier promotes the argument to an array of doubles.

Enhanced Modifiers to ANSI C Standards

Modifier	Supported with Format Code	Description
A comma (,) followed by an integer *n*, where *n* represents the array size.	%d and %f only	The corresponding argument is interpreted as a reference to the first element of an array of size *n*. The first *n* elements of this list are printed in the format specified by the format code. An asterisk (*) may be present after the comma (,) modifier, in which case an extra **arg** is used. This **arg** must be an integer representing the array size of the given type.
@1	%d and %f only	Converts to an IEEE 488.2 defined NR1 compatible number, which is an integer without any decimal point (for example, 123).
@2	%d and %f only	Converts to an IEEE 488.2 defined NR2 compatible number. The NR2 number has at least one digit after the decimal point (for example, 123.45).
@3	%d and %f only	Converts to an IEEE 488.2 defined NR3 compatible number. An NR3 number is a floating point number represented in an exponential form (for example, 1.2345E-67).
@H	%d and %f only	Converts to an IEEE 488.2 defined <HEXADECIMAL NUMERIC RESPONSE DATA>. The number is represented in a base of sixteen form. Only capital letters should represent numbers. The number is of form #H*XXX*.., where *XXX*.. is a hexadecimal number (for example, #HAF35B).

Modifier	Supported with Format Code	Description
@Q	%d and %f only	Converts to an IEEE 488.2 defined <OCTAL NUMERIC RESPONSE DATA>. The number is represented in a base of eight form. The number is of the form #Q*YYY.*., where *YYY.*. is an octal number (for example, #Q71234).
@B	%d and %f only	Converts to an IEEE 488.2 defined <BINARY NUMERIC RESPONSE DATA>. The number is represented in a base two form. The number is of the form #B*ZZZ.*., where *ZZZ.*. is a binary number (for example, #B011101001).

The following are the allowed format code characters. A format specifier sequence should include one and only one format code.

Standard ANSI C Format Codes

% Send the ASCII percent (%) character.

c Argument type: A character to be sent.

d Argument type: An integer.

Modifier	Interpretation
Default functionality	Print an integer in NR1 format (an integer without a decimal point).
@2 or @3	The integer is converted into a floating point number and output in the correct format.
field width	Minimum field width of the output number. Any of the six IEEE 488.2 modifiers can also be specified with *field width*.
Length modifier l	**arg** is a long integer.
Length modifier h	**arg** is a short integer.
, array size	**arg** points to an array of integers (or long or short integers, depending on the length modifier) of size array size. The elements of this array are separated by array size - 1 commas and output in the specified format.

f Argument type: A floating point number.

Modifier	Interpretation
Default functionality	Print a floating point number in NR2 format (a number with at least one digit after the decimal point).
@1	Print an integer in NR1 format. The number is truncated.
@3	Print a floating point number in NR3 format (scientific notation). *Precision* can also be specified.
field width	Minimum field width of the output number. Any of the six IEEE 488.2 modifiers can also be specified with *field width*.
Length modifier l	**arg** is a double float.
Length modifier L	**arg** is a long double.
, array size	**arg** points to an array of floats (or doubles or long doubles, depending on the length modifier) of size array size. The elements of this array are separated by array size - 1 commas and output in the specified format.

s Argument type: A reference to a NULL-terminated string that is sent to the device without change.

Enhanced Format Codes

b Argument type: A location of a block of data.

Flag or Modifier	Interpretation
Default functionality	The data block is sent as an IEEE 488.2 <DEFINITE LENGTH ARBITRARY BLOCK RESPONSE DATA>. A count (long integer) must appear as a flag that specifies the number of elements (by default, bytes) in the block. A *field width* or *precision* modifier is not allowed with this format code.
* (asterisk)	An asterisk may be present instead of the count. In such a case, two **args** are used, the first of which is a long integer specifying the count of the number of elements in the data block. The second **arg** is a reference to the data block. The size of an element is determined by the optional length modifier (see below), and the default is byte width.

Flag or Modifier	Interpretation
Length modifier h	**arg** points to an array of unsigned short integers (16 bits). The count corresponds to the number of words rather than bytes. The data is swapped and padded into standard IEEE 488.2 format, if native computer representation is different.
Length modifier l	**arg** points to an array of unsigned long integers. The count specifies the number of longwords (32 bits). Each longword data is swapped and padded into standard IEEE 488.2 format, if native computer representation is different.
Length modifier z	**arg** points to an array of floats. The count specifies the number of floating point numbers (32 bits). The numbers are represented in IEEE 754 format, if native computer representation is different.
Length modifier Z	**arg** points to an array of doubles. The count specifies the number of double floats (64 bits). The numbers will be represented in IEEE 754 format, if native computer representation is different.

B Argument type: A location of a block of data. The functionality is similar to **b,** except the data block is sent as an IEEE 488.2 <INDEFINITE LENGTH ARBITRARY BLOCK RESPONSE DATA>. This format involves sending an ASCII LF character with the END indicator set after the last byte of the block.

The END indicator is not appended when LF(\n) is part of a binary data block, as with %b or %B.

y Argument type: A location of a block of binary data.

Modifier	Interpretation
Default functionality	The data block is sent as raw binary data. A count (long integer) must appear as a flag that specifies the number of elements (by default, bytes) in the block. A field width or precision modifier is not allowed with this format code.
* (asterisk)	An asterisk may be present instead of the count. In such a case, two **arg**s are used, the first of which is a long integer specifying the count of the number of elements in the data block. The second **arg** is a reference to the data block. The size of an element is determined by the optional length modifier (see below), and the default is byte width.

Modifier	Interpretation
Length modifier h	**arg** points to an array of unsigned short integers (16 bits). The count corresponds to the number of words rather than bytes. If the optional !ol byte order modifier is present, the data is sent in little endian format; otherwise, the data is sent in standard IEEE 488.2 format. The data will be byte swapped and padded as appropriate if native computer representation is different.
Length modifier l	**arg** points to an array of unsigned long integers (32 bits). The count specifies the number of longwords rather than bytes. If the optional !ol byte order modifier is present, the data is sent in little endian format; otherwise, the data is sent in standard IEEE 488.2 format. The data will be byte swapped and padded as appropriate if native computer representation is different.
Byte order modifier !ob	Data is sent in standard IEEE 488.2 (big endian) format. This is the default behavior if neither !ob nor !ol is present.
Byte order modifier !ol	Data is sent in little endian format.

Other ANSI C Conversion Codes

For ANSI C compatibility, VISA also supports the following conversion codes for output codes: ‘i,’ ‘o,’ ‘u,’ ‘n,’ ‘x,’ ‘X,’ ‘e,’ ‘E,’ ‘g,’ ‘G’, and ‘p.’ For further explanation of these conversion codes, see the ANSI C Standard.

Related Items

See the `viSPrintf()`, `viVPrintf()`, and `viVSPrintf()` descriptions in this chapter. Also see the *INSTR Resource* description in Appendix C, *Resources.* Also refer to your ANSI C documentation for information on the `printf` function.

viQueryf

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

C Syntax

```
ViStatus viQueryf(ViSession vi, ViString writeFmt, ViString
                  readFmt,...)
```

Visual Basic Syntax

N/A

Purpose

Performs a formatted write and read through a single call to an operation.

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
writeFmt	IN	String describing the format of write arguments.
readFmt	IN	String describing the format of read arguments.
…	IN/OUT	Parameters to which write and read format strings are applied.

Return Values

Completion Codes	Description
VI_SUCCESS	Successfully completed the query operation.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_IO	Could not perform read/write operation because of I/O error.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.

Error Codes	Description
VI_ERROR_TMO	Timeout occurred before read/write operation completed.
VI_ERROR_INV_FMT	A format specifier in the **writeFmt** or **readFmt** string is invalid.
VI_ERROR_NSUP_FMT	The format specifier is not supported for current argument type.
VI_ERROR_ALLOC	The system could not allocate a formatted I/O buffer because of insufficient resources.

Description

This operation provides a mechanism of *Send, then receive* typical to a command sequence from a commander device. In this manner, the response generated from the command can be read immediately.

This operation is a combination of the `viPrintf()` and `viScanf()` operations. The first *n* arguments corresponding to the first format string are formatted by using the **writeFmt** string, then sent to the device. The write buffer is flushed immediately after the write portion of the operation completes. After these actions, the response data is read from the device into the remaining parameters (starting from parameter *n*+1) using the **readFmt** string.

Note ***Because the prototype for this function cannot provide complete type-checking, remember that all output parameters must be passed by reference.***

Related Items

See the `viPrintf()`, `viScanf()`, and `viVQueryf()` descriptions in this chapter. Also see the *INSTR Resource* description in Appendix C, *Resources*.

viRead

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

C Syntax

```
ViStatus viRead(ViSession vi, ViPBuf buf, ViUInt32 count,
                ViPUInt32 retCount)
```

Visual Basic Syntax

```
viRead&(ByVal vi&, ByVal buf$, ByVal count&, retCount&)
```

Purpose

Reads data from device synchronously.

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
buf	OUT	Location of a buffer to receive data from device.
count	IN	Number of bytes to be read.
retCount	OUT	Number of bytes actually transferred.

Return Values

Completion Codes	Description
VI_SUCCESS	The operation completed successfully and the END indicator was received (for interfaces that have END indicators). This completion code is returned regardless of whether the termination character is received or the number of bytes read is equal to **count**.

Completion Codes	Description
VI_SUCCESS_TERM_CHAR	The specified termination character was read but no END indicator was received. This completion code is returned regardless of whether the number of bytes read is equal to **count**.
VI_SUCCESS_MAX_CNT	The number of bytes read is equal to **count**. No END indicator was received and no termination character was read.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given **vi** does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_TMO	Timeout expired before operation completed.
VI_ERROR_RAW_WR_PROT_VIOL	Violation of raw write protocol occurred during transfer.
VI_ERROR_RAW_RD_PROT_VIOL	Violation of raw read protocol occurred during transfer.
VI_ERROR_OUTP_PROT_VIOL	Device reported an output protocol error during transfer.
VI_ERROR_BERR	Bus error occurred during transfer.
VI_ERROR_INV_SETUP	Unable to start read operation because setup is invalid (due to attributes being set to an inconsistent state).
VI_ERROR_NCIC	The interface associated with the given **vi** is not currently the controller in charge.
VI_ERROR_NLISTENERS	No-listeners condition is detected (both NRFD and NDAC are unasserted).
VI_ERROR_ASRL_PARITY	A parity error occurred during transfer.
VI_ERROR_ASRL_FRAMING	A framing error occurred during transfer.

Error Codes	Description
VI_ERROR_ASRL_OVERRUN	An overrun error occurred during transfer. A character was not read from the hardware before the next character arrived.
VI_ERROR_IO	An unknown I/O error occurred during transfer.

Description

The `viRead()` operation synchronously transfers data. The data read is to be stored in the buffer represented by **buf**. This operation returns only when the transfer terminates. Only one synchronous read operation can occur at any one time.

Related Items

See the `viReadAsync()`, `viBufRead()`, and `viWrite()` descriptions in this chapter. Also see the *INSTR Resource* description in Appendix C, *Resources*.

viReadAsync

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

C Syntax

```
ViStatus viReadAsync(ViSession vi, ViPBuf buf, ViUInt32 count,
                     ViPJobId jobId)
```

Visual Basic Syntax

N/A

Purpose

Reads data from device asynchronously.

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
buf	OUT	Location of a buffer to receive data from device.
count	IN	Number of bytes to be read.
jobId	OUT	Job ID of this asynchronous read operation.

Return Values

Completion Codes	Description
VI_SUCCESS	Asynchronous read operation successfully queued.
VI_SUCCESS_SYNC	Read operation performed synchronously.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_QUEUE_ERROR	Unable to queue read operation.

Description

The `viReadAsync()` operation asynchronously transfers data. The data read is to be stored in the buffer represented by **buf**. This operation normally returns before the transfer terminates.

Before calling this operation, you should enable the session for receiving I/O completion events. After the transfer has completed, an I/O completion event is posted.

The operation returns **jobId**, which you can use with either `viTerminate()` to abort the operation, or with an I/O completion event to identify which asynchronous read operation completed.

Related Items

See the `viEnableEvent()`, `viRead()`, `viTerminate()`, and `viWriteAsync()` descriptions in this chapter. See the `VI_EVENT_IO_COMPLETION` description in Chapter 4, *Events*. Also see the *INSTR Resource* description in Appendix C, *Resources*.

viReadSTB

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

C Syntax

```
ViStatus viReadSTB(ViSession vi, ViPUInt16 status)
```

Visual Basic Syntax

```
viReadSTB&(ByVal vi&, status%)
```

Purpose

Reads a status byte of the service request.

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
status	OUT	Service request status byte.

Return Values

Completion Codes	Description
VI_SUCCESS	The operation completed successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given **vi** does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_SRQ_NOCCURRED	Service request has not been received for the session.
VI_ERROR_TMO	Timeout expired before operation completed.

Error Codes	Description
VI_ERROR_RAW_WR_PROT_VIOL	Violation of raw write protocol occurred during transfer.
VI_ERROR_RAW_RD_PROT_VIOL	Violation of raw read protocol occurred during transfer.
VI_ERROR_BERR	Bus error occurred during transfer.
VI_ERROR_NCIC	The interface associated with the given **vi** is not currently the controller in charge.
VI_ERROR_NLISTENERS	No-listeners condition is detected (both NRFD and NDAC are unasserted).
VI_ERROR_INV_SETUP	Unable to start operation because setup is invalid (due to attributes being set to an inconsistent state).

Description

The viReadSTB() operation reads a service request status from a service requester (the message-based device). For example, on the IEEE 488.2 interface, the message is read by polling devices; for other types of interfaces, a message is sent in response to a service request to retrieve status information. For a serial device, if VI_ATTR_IO_PROT is VIASRL488, the device is sent the string "*STB?\n", and then the device's status byte is read; this operation is not valid for a serial device if VI_ATTR_IO_PROT is VI_NORMAL. If the status information is only one byte long, the most significant byte is returned with the zero value. If the service requester does not respond in the actual timeout period, VI_ERROR_TMO is returned.

Related Items

See the VI_ATTR_IO_PROT description in Chapter 3, *Attributes*. See the VI_EVENT_SERVICE_REQ description in Chapter 4, *Events*. Also see the *INSTR Resource* description in Appendix C, *Resources*.

viScanf

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

C Syntax

```
ViStatus viScanf(ViSession vi, ViString readFmt, ...)
```

Visual Basic Syntax

N/A

Purpose

Reads, converts, and formats data using the format specifier. Stores the formatted data in the parameters (designated by **...**).

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
readFmt	IN	String describing the format for arguments.
…	OUT	Parameters into which the data is read and the format string is applied.

Return Values

Completion Codes	Description
VI_SUCCESS	Data was successfully read and formatted into **...** parameter(s).

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_IO	Could not perform read operation because of I/O error.

Error Codes	Description
VI_ERROR_TMO	Timeout expired before read operation completed.
VI_ERROR_INV_FMT	A format specifier in the **readFmt** string is invalid.
VI_ERROR_NSUP_FMT	A format specifier in the **readFmt** string is not supported.
VI_ERROR_ALLOC	The system could not allocate a formatted I/O buffer because of insufficient resources.

Description

The viScanf() operation receives data from a device, formats it by using the format string, and stores the resulting data in the **arg** parameter list. The viRead() operation is used for the actual low-level read from the device. As a result, you should not use the viRead() and viScanf() operations in the same session.

☞ **Note** ***Because the prototype for this function cannot provide complete type-checking, remember that all output parameters must be passed by reference.***

The format string can have format specifier sequences, white characters, and ordinary characters. The white characters—blank, vertical tabs, horizontal tabs, form feeds, new line/linefeed, and carriage return—are ignored except in the case of %c and %[ ]. All other ordinary characters except % should match the next character read from the device.

The format string consists of a %, followed by optional modifier flags, followed by one of the format codes in that sequence. It is of the form

%[modifier]format code

where the optional modifier describes the data format, while format code indicates the nature of data (data type). One and only one format code should be performed at the specifier sequence. A format specification directs the conversion to the next input **arg**.

The results of the conversion are placed in the variable that the corresponding argument points to, unless the * assignment-suppressing character is given. In such a case, no **arg** is used and the results are ignored.

The viScanf() operation accepts input until an END indicator is read or all the format specifiers in the **readFmt** string are satisfied. Thus, detecting an END indicator before the **readFmt** string is fully consumed will result in ignoring the rest of the format string. Also, if some data remains in the buffer after all format specifiers in the **readFmt** string are satisfied, the data will be kept in the buffer and will be used by the next viScanf() operation.

When `viScanf()` times out, the next call to `viScanf()` will read from an empty buffer and force a read from the device.

Notice that when an END indicator is received, not all arguments in the format string may be consumed. However, the operation still returns a successful completion code.

The following two tables describe optional modifiers that can be used in a format specifier sequence.

ANSI C Standard Modifiers

Modifier	Supported with Format Code	Description
An integer representing the *field width*	%s, %c, %[ ] format codes	It specifies the maximum field width that the argument will take. A '#' may also appear instead of the integer *field width*, in which case the next **arg** is a reference to the *field width*. This **arg** is a reference to an integer for %c and %s. The *field width* is not allowed for %d or %f.
A length modifier ('h,' 'l,' 'L,' 'z,' or 'Z'). z and Z are not ANSI C standard modifiers.	h (d, b format codes) l (d, f, b format codes) L (f format code) z (b format code) Z (b format code)	The argument length modifiers specify one of the following: a. The h modifier promotes the argument to be a reference to a short integer or unsigned short integer, depending on the format code. b. The l modifier promotes the argument to point to a long integer or unsigned long integer. c. The L modifier promotes the argument to point to a long double floats parameter. d. The z modifier promotes the argument to point to an array of floats. e. The Z modifier promotes the argument to point to an array of double floats.
*	All format codes	An asterisk (*) acts as the assignment suppression character. The input is not assigned to any parameters and is discarded.

Enhanced Modifiers to ANSI C Standards

Modifier	Supported with Format Code	Description
A comma (,) followed by an integer *n*, where *n* represents the array size.	%d and %f only	The corresponding argument is interpreted as a reference to the first element of an array of size *n*. The first *n* elements of this list are printed in the format specified by the format code. A number sign (#) may be present after the comma (,) modifier, in which case an extra **arg** is used. This **arg** must be an integer representing the array size of the given type.

Format Codes

ANSI C Format Codes

c Argument type: A reference to a character.

Flags or Modifiers	Interpretation
Default functionality	A character is read from the device and stored in the parameter.
field width	*field width* number of characters are read and stored at the reference location (the default *field width* is 1). No NULL character is added at the end of the data block.

☞ **Note** ***This format code does not ignore white space in the device input stream.***

d Argument type: A reference to an integer.

Flags or Modifiers	Interpretation
Default functionality	Characters are read from the device until an entire number is read. The number read may be in either IEEE 488.2 formats <DECIMAL NUMERIC PROGRAM DATA>, also known as NRf; flexible numeric representation (NR1, NR2, NR3...); or <NON-DECIMAL NUMERIC PROGRAM DATA> (#H, #Q, and #B).
field width	The input number will be stored in a field at least this wide.

Flags or Modifiers	Interpretation
Length modifier l	**arg** is a reference to a long integer.
Length modifier h	**arg** is a reference to a short integer. Rounding is performed according to IEEE 488.2 rules (0.5 and up).
, array size	**arg** points to an array of integers (or long or short integers, depending on the length modifier) of size array size. The elements of this array should be separated by commas. Elements will be read until either array size number of elements are consumed or they are no longer separated by commas.

f Argument type: A reference to a floating point number.

Flags or Modifiers	Interpretation
Default functionality	Characters are read from the device until an entire number is read. The number read may be in either IEEE 488.2 formats <DECIMAL NUMERIC PROGRAM DATA> (NRf) or <NON-DECIMAL NUMERIC PROGRAM DATA> (#H, #Q, and #B)
field width	The input will be stored in a field at least this wide.
Length modifier l	**arg** is a reference to a double floating point number.
Length modifier L	**arg** is a reference to a long double number.
, array size	**arg** points to an array of floats (or double or long double, depending on the length modifier) of size array size. The elements of this array should be separated by commas. Elements will be read until either array size number of elements are consumed or they are no longer separated by commas.

s Argument type: A reference to a string.

Flags or Modifiers	Interpretation
Default functionality	All leading white space characters are ignored. Characters are read from the device into the string until a white space character is read.
field width	This flag gives the maximum string size. If the *field width* contains a number sign (#), two arguments are used. The first argument read is a pointer to an integer specifying the maximum array size. The second should be a reference to an array. In case of *field width* characters already read before encountering a white space, additional characters are read and discarded until a white space character is found. In case of # *field width*, the actual number of characters read are stored back in the integer pointed to by the first argument.

Enhanced Format Codes

b Argument type: A reference to a data array.

Flags or Modifiers	Interpretation
Default functionality	The data must be in IEEE 488.2 <ARBITRARY BLOCK PROGRAM DATA> format. The format specifier sequence should have a flag describing the *field width*, which will give a maximum count of the number of bytes (or words or longwords, depending on length modifiers) to be read from the device. If the *field width* contains a # sign, two arguments are used. The first **arg** read is a pointer to a long integer specifying the maximum number of elements that the array can hold. The second **arg** should be a reference to an array. Also, the actual number of elements read is stored back in the first argument. In absence of length modifiers, the data is assumed to be of byte-size elements. In some cases, data might be read until an END indicator is read.
Length modifier h	**arg** points to an array of 16-bit words, and count specifies the number of words. Data that is read is assumed to be in IEEE 488.2 byte ordering. It will be byte swapped and padded as appropriate to native computer format.

Flags or Modifiers	Interpretation
Length modifier l	**arg** points to an array of 32-bit longwords, and count specifies the number of longwords. Data that is read is assumed to be in IEEE 488.2 byte ordering. It will be byte swapped and padded as appropriate to native computer format.
Length modifier z	**arg** points to an array of floats, and count specifies the number of floating point numbers. Data that is read is an array of 32-bit IEEE 754 format floating point numbers.
Length modifier Z	**arg** points to an array of doubles, and the count specifies the number of floating point numbers. Data that is read is an array of 64-bit IEEE 754 format floating point numbers.

t Argument type: A reference to a string.

Flags or Modifiers	Interpretation
Default functionality	Characters are read from the device until the first END indicator is received. The character on which the END indicator was received is included in the buffer.
field width	This flag gives the maximum string size. If an END indicator is not received before *field width* number of characters, additional characters are read and discarded until an END indicator arrives. *#field width* has the same meaning as in %s.

T Argument type: A reference to a string.

Flags or Modifiers	Interpretation
Default functionality	Characters are read from the device until the first linefeed character (\n) is received. The linefeed character is included in the buffer.
field width	This flag gives the maximum string size. If a linefeed character is not received before *field width* number of characters, additional characters are read and discarded until a linefeed character arrives. *#field width* has the same meaning as in %s.

y Argument type: A location of a block of binary data.

Modifier	Interpretation
Default functionality	The data block is read as raw binary data. The format specifier sequence should have a flag describing the array size, which will give a maximum count of the number of bytes (or words or longwords, depending on length modifiers) to be read from the device. If the array size contains a # sign, two arguments are used. The first argument read is a pointer to a long integer that specifies the maximum number of elements that the array can hold. The second argument should be a reference to an array. Also, the actual number of elements read is stored back in the first argument. In absence of length modifiers, the data is assumed to be byte-size elements. In some cases, data might be read until an END indicator is read.
Length modifier h	The data block is assumed to be a reference to an array of unsigned short integers (16 bits). The count corresponds to the number of words rather than bytes. If the optional “!ol” modifier is present, the data read is assumed to be in little endian format; otherwise, the data read is assumed to be in standard IEEE 488.2 format. The data will be byte swapped and padded as appropriate to native computer format.
Length modifier l	The data block is assumed to be a reference to an array of unsigned long integers (32 bits). The count corresponds to the number of longwords rather than bytes. If the optional “!ol” modifier is present, the data read is assumed to be in little endian format; otherwise, the data read is assumed to be in standard IEEE 488.2 format. The data will be byte swapped and padded as appropriate to native computer format.
Byte order modifier !ob	The data being read is assumed to be in standard IEEE 488.2 (big endian) format. This is the default behavior if neither !ob nor !ol is present.
Byte order modifier !ol	The data being read is assumed to be in little endian format.

Other ANSI C Format Specifiers

For ANSI C compatibility, VISA also supports the following format specifiers for input codes: ‘i,’ ‘o,’ ‘u,’ ‘n,’ ‘x,’ ‘X,’ ‘e,’ ‘E,’ ‘g,’ ‘G,’ ‘p,’ ‘[...],’ and ‘[^...].’ For further explanation of these conversion codes, see the ANSI C Standard.

Related Items

See the viSScanf(), viVScanf(), and viVSScanf() descriptions in this chapter. Also see the *INSTR Resource* description in Appendix C, *Resources*. Also refer to your ANSI C documentation for information on the scanf function.

viSetAttribute

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

C Syntax

```
ViStatus viSetAttribute(ViObject vi, ViAttr attribute,
                        ViAttrState attrState)
```

Visual Basic Syntax

```
viSetAttribute&(ByVal vi&, ByVal attribute&, ByVal attrState&)
```

Purpose

Sets the state of an attribute.

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
attribute	IN	Attribute for which the state is to be modified.
attrState	IN	The state of the attribute to be set for the specified object. The interpretation of the individual attribute value is defined by the object.

Return Values

Completion Codes	Description
VI_SUCCESS	Attribute value set successfully.
VI_WARN_NSUP_ATTR_STATE	Although the specified attribute state is valid, it is not supported by this implementation.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given object reference is invalid.
VI_ERROR_NSUP_ATTR	The specified attribute is not defined by the referenced object.

Error Codes	Description
VI_ERROR_NSUP_ATTR_STATE	The specified state of the attribute is not valid, or is not supported as defined by the object.
VI_ERROR_ATTR_READONLY	The specified attribute is read-only.

Description

The viSetAttribute() operation is used to modify the state of an attribute for the specified object.

Both VI_WARN_NSUP_ATTR_STATE and VI_ERROR_NSUP_ATTR_STATE indicate that the specified attribute state is not supported. A resource normally returns the error code VI_ERROR_NSUP_ATTR_STATE when it cannot set a specified attribute state. The completion code VI_WARN_NSUP_ATTR_STATE is intended to alert the application that although the specified optional attribute state is not supported, the application should not fail. One example is attempting to set an attribute value that would increase performance speeds. This is different than attempting to set an attribute value that specifies required but nonexistent hardware (such as specifying a VXI ECL trigger line when no hardware support exists) or a value that would change assumptions a resource might make about the way data is stored or formatted (such as byte order).

Related Items

See the viGetAttribute() description in this chapter. Also see the *VISA Resource Template* description in Appendix C, *Resources*, and the attribute descriptions in Chapter 3, *Attributes*.

viSetBuf

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

C Syntax

```
ViStatus viSetBuf(ViSession vi, ViUInt16 mask, ViUInt32 size)
```

Visual Basic Syntax

```
viSetBuf&(ByVal vi&, ByVal mask%, ByVal size&)
```

Purpose

Sets the size for the formatted I/O and/or serial communication buffer(s).

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
mask	IN	Specifies the type of buffer.
size	IN	The size to be set for the specified buffer(s).

Return Values

Completion Codes	Description
VI_SUCCESS	Buffer size set successfully.
VI_WARN_NSUP_BUF	The specified buffer is not supported.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.

Error Codes	Description
VI_ERROR_ALLOC	The system could not allocate the buffer(s) of the specified **size** because of insufficient resources.
VI_ERROR_INV_MASK	The system cannot set the buffer for the given **mask**.

Description

The viSetBuf() operation changes the buffer size of the read and/or write buffer for formatted I/O and/or serial communication. The **mask** parameter specifies the buffer for which to set the size. The **mask** parameter can specify multiple buffers by bit-ORing any of the following values together.

Flags	Interpretation
VI_READ_BUF (1)	Formatted I/O read buffer.
VI_WRITE_BUF (2)	Formatted I/O write buffer.
VI_ASRL_IN_BUF (16)	Serial communication receive buffer.
VI_ASRL_OUT_BUF (32)	Serial communication transmit buffer.

A call to viSetBuf() flushes the session's related read/write buffer(s). Although you can explicitly flush the buffers by making a call to viFlush(), the buffers are flushed implicitly under some conditions. These conditions vary for the viPrintf() and viScanf() operations.

Since not all serial drivers support user-defined buffer sizes, it is possible that a specific implementation of VISA may not be able to control this feature. If an application requires a specific buffer size for performance reasons, but a specific implementation of VISA cannot guarantee that size, then it is recommended to use some form of handshaking to prevent overflow conditions.

Related Items

See the viFlush(), viPrintf(), and viScanf() descriptions in this chapter. Also see the *INSTR Resource* description in Appendix C, *Resources*.

viSPrintf

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

C Syntax

```
ViStatus viSPrintf(ViSession vi, ViPBuf buf, ViString writeFmt, ...)
```

Visual Basic Syntax

N/A

Purpose

Converts, formats, and sends the parameters (designated by**...**) to a user-specified buffer as specified by the format string.

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
buf	OUT	Buffer where data is to be written.
writeFmt	IN	The format string to apply to parameters in `ViVAList`.
...	IN	Parameters to which the format string is applied. The formatted data is written to the specified **buf**.

Return Values

Completion Codes	Description
`VI_SUCCESS`	Parameters were successfully formatted.

Error Codes	Description
`VI_ERROR_INV_OBJECT`	The given session reference is invalid.
`VI_ERROR_RSRC_LOCKED`	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
`VI_ERROR_INV_FMT`	A format specifier in the **writeFmt** string is invalid.

Error Codes	Description
VI_ERROR_NSUP_FMT	A format specifier in the **writeFmt** string is not supported.
VI_ERROR_ALLOC	The system could not allocate a formatted I/O buffer because of insufficient resources.

Description

The `viSPrintf()` operation is similar to `viPrintf()`, except that the output is not written to the device; it is written to the user-specified buffer. This output buffer will be NULL terminated.

If this operation outputs an END indicator before all the arguments are satisfied, then the rest of the **writeFmt** string is ignored and the buffer string is still terminated by a NULL.

Related Items

See the `viPrintf()`, `viVPrintf()`, and `viVSPrintf()` descriptions in this chapter. Also see the *INSTR Resource* description in Appendix C, *Resources*.

viSScanf

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

C Syntax

```
ViStatus viSScanf(ViSession vi, ViBuf buf, ViString readFmt, ...)
```

Visual Basic Syntax

N/A

Purpose

Reads, converts, and formats data from a user-specified buffer using the format specifier. Stores the formatted data in the parameters (designated by **...**).

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
buf	IN	Buffer from which data is read and formatted.
readFmt	IN	String describing the format for arguments.
...	OUT	Parameters into which the data is read and the format string is applied.

Return Values

Completion Codes	Description
VI_SUCCESS	Data was successfully read and formatted into **...** parameter(s).

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_INV_FMT	A format specifier in the **readFmt** string is invalid.

Error Codes	Description
VI_ERROR_NSUP_FMT	A format specifier in the **readFmt** string is not supported.
VI_ERROR_ALLOC	The system could not allocate a formatted I/O buffer because of insufficient resources.

Description

The viSScanf() operation is similar to viScanf(), except that the data is read from a user-specified buffer rather than from a device.

Related Items

See the viScanf(), viVScanf(), and viVSScanf() descriptions in this chapter. Also see the *INSTR Resource* description in Appendix C, *Resources*.

viStatusDesc

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

C Syntax

```
ViStatus viStatusDesc(ViObject vi, ViStatus status, ViChar desc[])
```

Visual Basic Syntax

```
viStatusDesc&(ByVal vi&, ByVal status&, ByVal desc$)
```

Purpose

Returns a user-readable description of the status code passed to the operation.

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
status	IN	Status code to interpret.
desc	OUT	The user-readable string interpretation of the status code passed to the operation.

Return Values

Completion Codes	Description
VI_SUCCESS	Description successfully returned.
VI_WARN_UNKNOWN_STATUS	The status code passed to the operation could not be interpreted.

Description

The `viStatusDesc()` operation is used to retrieve a user-readable string that describes the status code presented. If the string cannot be interpreted, the operation returns the warning code `VI_WARN_UNKNOWN_STATUS`. However, the output string **desc** is valid regardless of the status return value.

☞ **Note** ***The size of the desc parameter should be at least 256 bytes.***

Related Items

See Appendix B, *Status Codes*, for a complete list of the possible status codes for each operation. Also see the *VISA Resource Template* description in Appendix C, *Resources*.

viTerminate

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

C Syntax

```
ViStatus viTerminate(ViObject vi, ViUInt16 degree, ViJobId jobId))
```

Visual Basic Syntax

N/A

Purpose

Requests a VISA session to terminate normal execution of an asynchronous operation.

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
degree	IN	VI_NULL (0).
jobId	IN	Specifies an operation identifier.

Return Values

Completion Codes	Description
VI_SUCCESS	Request serviced successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given object reference is invalid.
VI_ERROR_INV_JOB_ID	Specified job identifier is invalid.
VI_ERROR_INV_DEGREE	Specified **degree** is invalid.

Description

This operation is used to request a session to terminate normal execution of an operation, as specified by the **jobId** parameter. The **jobId** parameter is a unique value generated from each call to an asynchronous operation.

If the viTerminate() operation causes the specified asynchronous operation to be aborted, the resulting I/O completion event contains the status code VI_ERROR_ABORT. If the operation associated with the specified **jobId** has already completed, the viTerminate() operation returns VI_ERROR_INV_JOB_ID.

Related Items

See the viReadAsync() and viWriteAsync() descriptions in this chapter. See the VI_EVENT_IO_COMPLETION description in Chapter 4, *Events*. Also see the *VISA Resource Template* description in Appendix C, *Resources*.

viUninstallHandler

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

C Syntax

```
ViStatus viUninstallHandler(ViSession vi, ViEventType eventType,
                            ViHndlr handler, ViAddr userHandle)
```

Visual Basic Syntax

N/A

Purpose

Uninstalls handlers for events.

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
eventType	IN	Logical event identifier.
handler	IN	Interpreted as a valid reference to a handler to be uninstalled by a client application.
userHandle	IN	A value specified by an application that can be used for identifying handlers uniquely in a session for an event.

Return Values

Completion Codes	Description
VI_SUCCESS	Event handler successfully uninstalled.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_INV_EVENT	Specified event type is not supported by the resource.

Error Codes	Description
VI_ERROR_INV_HNDLR_REF	Either the specified handler reference or the user context value (or both) does not match any installed handler.
VI_ERROR_HNDLR_NINSTALLED	A handler is not currently installed for the specified event.

Description

The viUninstallHandler() operation allows applications to uninstall handlers for events on sessions. Applications should also specify the value in the **userHandle** parameter that was passed while installing the handler. VISA identifies handlers uniquely using the handler reference and this value. All the handlers, for which the handler reference and the value matches, are uninstalled. Specifying VI_ANY_HNDLR as the value for the **handler** parameter causes the operation to uninstall all the handlers with the matching value in the **userHandle** parameter.

Related Items

See the viInstallHandler() description in this chapter. Also see the viEventHandler() description for its parameter description. Also see the *VISA Resource Template* description in Appendix C, *Resources*.

viUnlock

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

C Syntax

```
ViStatus viUnlock(ViSession vi)
```

Visual Basic Syntax

```
viUnlock&(ByVal vi&)
```

Purpose

Relinquishes a lock for the specified resource.

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.

Return Values

Completion Codes	Description
VI_SUCCESS	Lock successfully relinquished.
VI_SUCCESS_NESTED_EXCLUSIVE	Call succeeded, but this session still has nested exclusive locks.
VI_SUCCESS_NESTED_SHARED	Call succeeded, but this session still has nested shared locks.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_SESN_NLOCKED	The current session did not have any lock on the resource.

Description

This operation is used to relinquish the lock previously obtained using the `viLock()` operation.

Related Items

See the viLock() description in this chapter. Also see the *VISA Resource Template* description in Appendix C, *Resources*.

viUnmapAddress

❑ Serial	❑ GPIB	■ GPIB-VXI	■ VXI

C Syntax

```
ViStatus viUnmapAddress(ViSession vi)
```

Visual Basic Syntax

```
viUnmapAddress&(ByVal vi&)
```

Purpose

Unmaps memory space previously mapped by `viMapAddress()`.

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.

Return Values

Completion Codes	Description
VI_SUCCESS	Operation completed successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given **vi** does not support this operation.
VI_ERROR_WINDOW_NMAPPED	The specified session is not currently mapped.

Description

The `viUnmapAddress()` operation unmaps the region previously mapped by the `viMapAddress()` operation for this session.

Related Items

See the `viMapAddress()` description in this chapter. Also see the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix C, *Resources*.

viVPrintf

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

C Syntax

```
ViStatus viVPrintf(ViSession vi, ViString writeFmt,
                   ViVAList params)
```

Visual Basic Syntax

```
viVPrintf&(ByVal vi&, ByVal writeFmt$, params as Any)
```

Purpose

Converts, formats, and sends the parameters designated by **params** to the device as specified by the format string.

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
writeFmt	IN	String describing the format to apply to **params**.
params	IN	A list containing the variable number of parameters on which the format string is applied. The formatted data is written to the specified device.

Return Values

Completion Codes	Description
VI_SUCCESS	Parameters were successfully formatted.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_IO	Could not perform write operation because of I/O error.

Error Codes	Description
VI_ERROR_TMO	Timeout expired before write operation completed.
VI_ERROR_INV_FMT	A format specifier in the **writeFmt** string is invalid.
VI_ERROR_NSUP_FMT	A format specifier in the **writeFmt** string is not supported.
VI_ERROR_ALLOC	The system could not allocate a formatted I/O buffer because of insufficient resources.

Description

This operation is similar to `viPrintf()`, except that the **params** parameters list provides the parameters rather than separate **arg** parameters.

Related Items

See the `viPrintf()`, `viSPrintf()`, and `viVSPrintf()` descriptions in this chapter. Also see the *INSTR Resource* description in Appendix C, *Resources*.

viVQueryf

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

C Syntax

```
ViStatus viVQueryf(ViSession vi, ViString writeFmt, ViString
                   readFmt, ViVAList params)
```

Visual Basic Syntax

```
viVQueryf&(ByVal vi&, ByVal writeFmt$, ByVal readFmt$,
           params as Any)
```

Purpose

Performs a formatted write and read through a single call to an operation.

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
writeFmt	IN	String describing the format of write arguments.
readFmt	IN	String describing the format of read arguments.
params	IN/OUT	A list containing the variable number of write and read parameters. The write parameters are formatted and written to the specified device. The read parameters store the data read from the device after the format string is applied to the data.

Return Values

Completion Codes	Description
VI_SUCCESS	Successfully completed the query operation.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_IO	Could not perform read/write operation because of I/O error.
VI_ERROR_TMO	Timeout occurred before read/write operation completed.
VI_ERROR_INV_FMT	A format specifier in the **writeFmt** or **readFmt** string is invalid.
VI_ERROR_NSUP_FMT	The format specifier is not supported for current argument type.
VI_ERROR_ALLOC	The system could not allocate a formatted I/O buffer because of insufficient resources.

Description

This operation is similar to `viQueryf()`, except that the **params** parameters list provides the parameters rather than the separate **arg** parameter list

☞ **Note** ***Because the prototype for this function cannot provide complete type-checking, remember that all output parameters must be passed by reference.***

Related Items

See the `viQueryf()` description in this chapter. Also see the *INSTR Resource* description in Appendix C, *Resources*.

viVScanf

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

C Syntax

```
ViStatus viVScanf(ViSession vi, ViString readFmt, ViVAList params)
```

Visual Basic Syntax

```
viVScanf&(ByVal vi&, ByVal readFmt$, params as Any)
```

Purpose

Reads, converts, and formats data using the format specifier. Stores the formatted data in the parameters designated by **params**.

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
readFmt	IN	String describing the format to apply to **params**.
params	OUT	A list with the variable number of parameters into which the data is read and the format string is applied.

Return Values

Completion Codes	Description
VI_SUCCESS	Data was successfully read and formatted into **params** parameter(s).

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_IO	Could not perform read operation because of I/O error.

Error Codes	Description
VI_ERROR_TMO	Timeout expired before read operation completed.
VI_ERROR_INV_FMT	A format specifier in the **readFmt** string is invalid.
VI_ERROR_NSUP_FMT	A format specifier in the **readFmt** string is not supported.
VI_ERROR_ALLOC	The system could not allocate a formatted I/O buffer because of insufficient resources.

Description

This operation is similar to `viScanf()`, except that the **params** parameters list provides the parameters rather than separate **arg** parameters.

☞ **Note** ***Because the prototype for this function cannot provide complete type-checking, remember that all output parameters must be passed by reference.***

Related Items

See the `viScanf()`, `viSScanf()`, and `viVSScanf()` descriptions in this chapter. Also see the *INSTR Resource* description in Appendix C, *Resources*.

viVSPrintf

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

C Syntax

```
ViStatus viVSPrintf(ViSession vi, ViPBuf buf, ViString writeFmt,
                    ViVAList params)
```

Visual Basic Syntax

```
viVSPrintf&(ByVal vi&, ByVal buf$, ByVal writeFmt$, params as Any)
```

Purpose

Converts, formats, and sends the parameters designated by **params** to a user-specified buffer as specified by the format string.

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
buf	OUT	Buffer where data is to be written.
writeFmt	IN	The format string to apply to parameters in `ViVAList`.
params	IN	A list containing the variable number of parameters on which the format string is applied. The formatted data is written to the specified **buf**.

Return Values

Completion Codes	Description
VI_SUCCESS	Parameters were successfully formatted.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.

Error Codes	Description
VI_ERROR_INV_FMT	A format specifier in the **writeFmt** string is invalid.
VI_ERROR_NSUP_FMT	A format specifier in the **writeFmt** string is not supported.
VI_ERROR_ALLOC	The system could not allocate a formatted I/O buffer because of insufficient resources.

Description

This operation is similar to `viVPrintf()`, except that the output is not written to the device; it is written to the user-specified buffer. This output buffer is NULL terminated.

If this operation outputs an END indicator before all the arguments are satisfied, then the rest of the **writeFmt** string is ignored and the buffer string is still terminated by a NULL.

Related Items

See the `viPrintf()`, `viSPrintf()`, and `viVPrintf()` descriptions in this chapter. Also see the *INSTR Resource* description in Appendix C, *Resources*.

viVSScanf

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

C Syntax

```
ViStatus viVSScanf(ViSession vi, ViBuf buf, ViString readFmt,
                   ViVAList params)
```

Visual Basic Syntax

```
viVSScanf&(ByVal vi&, ByVal buf$, ByVal readFmt$, params as Any)
```

Purpose

Reads, converts, and formats data from a user-specified buffer using the format specifier. Stores the formatted data in the parameters designated by **params**.

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
buf	IN	Buffer from which data is read and formatted.
readFmt	IN	String describing the format to apply to **params**.
params	OUT	A list with the variable number of parameters into which the data is read and the format string is applied.

Return Values

Completion Codes	Description
VI_SUCCESS	Data was successfully read and formatted into **params** parameter(s).

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.

Error Codes	Description
VI_ERROR_INV_FMT	A format specifier in the **readFmt** string is invalid.
VI_ERROR_NSUP_FMT	A format specifier in the **readFmt** string is not supported.
VI_ERROR_ALLOC	The system could not allocate a formatted I/O buffer because of insufficient resources.

Description

The viVSScanf() operation is similar to viVScanf(), except that the data is read from a user-specified buffer rather than a device.

Note ***Because the prototype for this function cannot provide complete type checking, remember that all output parameters must be passed by reference.***

Related Items

See the viScanf(), viSScanf(), and viVScanf() descriptions in this chapter. Also see the *INSTR Resource* description in Appendix C, *Resources*.

viVxiCommandQuery

❑ Serial	❑ GPIB	■ GPIB-VXI	■ VXI

C Syntax

```
ViStatus viVxiCommandQuery(ViSession vi, ViUInt16 mode,
                           ViUInt32 cmd, ViPUInt32 response)
```

Visual Basic Syntax

```
viVxiCommandQuery&(ByVal vi&, ByVal mode%, ByVal cmd&, response&)
```

Purpose

Sends the device a miscellaneous command or query and/or retrieves the response to a previous query.

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
mode	IN	Specifies whether to issue a command and/or retrieve a response. See the *Description* section for actual values.
cmd	IN	The miscellaneous command to send.
response	OUT	The response retrieved from the device. If the mode specifies to send a command rather than retrieve a response, you can use `VI_NULL` for this parameter.

Return Values

Completion Codes	Description
`VI_SUCCESS`	The operation completed successfully.

Error Codes	Description
`VI_ERROR_INV_OBJECT`	The given session reference is invalid.
`VI_ERROR_NSUP_OPER`	The given **vi** does not support this operation.

Error Codes	Description
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_TMO	Timeout expired before operation completed.
VI_ERROR_RAW_WR_PROT_VIOL	Violation of raw write protocol occurred during transfer.
VI_ERROR_RAW_RD_PROT_VIOL	Violation of raw read protocol occurred during transfer.
VI_ERROR_OUTP_PROT_VIOL	Device reported an output protocol error during transfer.
VI_ERROR_INP_PROT_VIOL	Device reported an input protocol error during transfer.
VI_ERROR_BERR	Bus error occurred during transfer.
VI_ERROR_RESP_PENDING	A previous response is still pending, causing a multiple query error.
VI_ERROR_INV_MODE	The value specified by the **mode** parameter is invalid.

Description

The viVxiCommandQuery() operation can send a command or query, or receive a response to a query previously sent to the device. The **mode** parameter specifies whether to issue a command and/or retrieve a response, and indicates the type or size of command and/or response to use. The following table defines the values for the **mode** parameter.

Mode	Action Description
VI_VXI_CMD16	Send 16-bit Word Serial command.
VI_VXI_CMD16_RESP16	Send 16-bit Word Serial query; get 16-bit response.
VI_VXI_RESP16	Get 16-bit response from previous query.
VI_VXI_CMD32	Send 32-bit Word Serial command.
VI_VXI_CMD32_RESP16	Send 32-bit Word Serial query; get 16-bit response.
VI_VXI_CMD32_RESP32	Send 32-bit Word Serial query; get 32-bit response.
VI_VXI_RESP32	Get 32-bit response from previous query.

Notice that the **mode** you specify can cause all or part of the **cmd** or **response** parameters to be ignored.

- If **mode** specifies sending a 16-bit command, the upper half of **cmd** is ignored.
- If **mode** specifies retrieving a response only, **cmd** is ignored.
- If **mode** specifies sending a command only, **response** is ignored. You can use `VI_NULL` for the value of **response**.
- If **mode** specifies to retrieve a 16-bit value, the upper half of **response** is set to 0.

Related Items

See the *INSTR Resource* description in Appendix C, *Resources*. Also refer to the *VXI Specification* for defined Word Serial commands.

viWaitOnEvent

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

C Syntax

```
ViStatus viWaitOnEvent(ViSession vi, ViEventType inEventType,
                       ViUInt32 timeout, ViPEventType
                       outEventType, ViPEvent outContext)
```

Visual Basic Syntax

```
viWaitOnEvent&(ByVal vi&, ByVal inEventType&, ByVal timeout&,
               outEventType&, outContext&)
```

Purpose

Waits for an occurrence of the specified event for a given session.

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
inEventType	IN	Logical identifier of the event(s) to wait for.
timeout	IN	Absolute time period in time units that the resource shall wait for a specified event to occur before returning the time elapsed error. The time unit is in milliseconds.
outEventType	OUT	Logical identifier of the event actually received.
outContext	OUT	A handle specifying the unique occurrence of an event.

Return Values

Completion Codes	Description
VI_SUCCESS	Wait terminated successfully on receipt of an event occurrence. The queue is empty.
VI_SUCCESS_QUEUE_NEMPTY	Wait terminated successfully on receipt of an event notification. There is still at least one more event occurrence of the type specified by **inEventType** available for this session.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_INV_EVENT	Specified event type is not supported by the resource.
VI_ERROR_TMO	Specified event did not occur within the specified time period.
VI_ERROR_NENABLED	The session must be enabled for events of the specified type in order to receive them.

Description

The viWaitOnEvent() operation suspends the execution of a thread of an application and waits for an event of the type specified by **inEventType** for a time period specified by **timeout**. You can wait only for events that have been enabled with the viEnableEvent() operation. Refer to individual event descriptions for context definitions. If the specified **inEventType** is VI_ALL_ENABLED_EVENTS, the operation waits for any event that is enabled for the given session. If the specified timeout value is VI_TMO_INFINITE, the operation is suspended indefinitely. If the specified timeout value is VI_TMO_IMMEDIATE, the operation is not suspended; therefore, this value can be used to dequeue events from an event queue.

When the **outContext** handle returned from a successful invocation of viWaitOnEvent() is no longer needed, it should be passed to viClose().

If a session's event queue becomes full and a new event arrives, the new event is discarded. The default event queue size (per session) is 50, which is sufficiently large for most applications. If an application expects more than 50 events to arrive without having been handled, it can modify the value of the attribute VI_ATTR_MAX_QUEUE_LENGTH to the required size.

The **outEventType** and **outContext** parameters are optional and can be VI_NULL. This can be used if the event type is known from the **inEventType** parameter, or if the **outContext** handle is not needed to retrieve additional information. If VI_NULL is used for the **outContext** parameter, VISA will automatically close the event context.

Related Items

See the viEnableEvent() and viClose() descriptions in this chapter. See the VI_ATTR_MAX_QUEUE_LENGTH description in Chapter 3, *Attributes*. See Chapter 4, *Events*, for a list of events that you can wait for. Also see the *VISA Resource Template*, *INSTR Resource*, and *MEMACC Resource* descriptions in Appendix C, *Resources*.

viWrite

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

C Syntax

```
ViStatus viWrite(ViSession vi, ViBuf buf, ViUInt32 count,
                 ViPUInt32 retCount)
```

Visual Basic Syntax

```
viWrite&(ByVal vi&, ByVal buf$, ByVal count&, retCount&)
```

Purpose

Writes data to device synchronously.

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
buf	IN	Location of a data block to be sent to a device.
count	IN	Number of bytes to be written.
retCount	OUT	Number of bytes actually transferred.

Return Values

Completion Codes	Description
VI_SUCCESS	Transfer completed.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given **vi** does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_TMO	Timeout expired before operation completed.

Error Codes	Description
VI_ERROR_RAW_WR_PROT_VIOL	Violation of raw write protocol occurred during transfer.
VI_ERROR_RAW_RD_PROT_VIOL	Violation of raw read protocol occurred during transfer.
VI_ERROR_INP_PROT_VIOL	Device reported an input protocol error during transfer.
VI_ERROR_BERR	Bus error occurred during transfer.
VI_ERROR_INV_SETUP	Unable to start write operation because setup is invalid (due to attributes being set to an inconsistent state).
VI_ERROR_NCIC	The interface associated with the given **vi** is not currently the controller in charge.
VI_ERROR_NLISTENERS	No-listeners condition is detected (both NRFD and NDAC are unasserted).
VI_ERROR_IO	An unknown I/O error occurred during transfer.

Description

The viWrite() operation synchronously transfers data. The data to be written is in the buffer represented by **buf**. This operation returns only when the transfer terminates. Only one synchronous write operation can occur at any one time.

Related Items

See the viRead(), viBufWrite(), and viWriteAsync() descriptions in this chapter. Also see the *INSTR Resource* description in Appendix C, *Resources*.

viWriteAsync

■ Serial	■ GPIB	■ GPIB-VXI	■ VXI

C Syntax

```
ViStatus viWriteAsync(ViSession vi, ViBuf buf, ViUInt32 count,
                      ViPJobId jobId)
```

Visual Basic Syntax

N/A

Purpose

Writes data to device asynchronously.

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
buf	IN	Location of a data block to be sent to a device.
count	IN	Number of bytes to be written.
jobId	OUT	Job ID of this asynchronous write operation.

Return Values

Completion Codes	Description
VI_SUCCESS	Asynchronous write operation successfully queued.
VI_SUCCESS_SYNC	Write operation performed synchronously.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_QUEUE_ERROR	Unable to queue write operation.

Description

The `viWriteAsync()` operation asynchronously transfers data. The data to be written is in the buffer represented by **buf**. This operation normally returns before the transfer terminates.

Before calling this operation, you should enable the session for receiving I/O completion events. After the transfer has completed, an I/O completion event is posted.

The operation returns a job identifier that you can use with either `viTerminate()` to abort the operation or with an I/O completion event to identify which asynchronous write operation completed.

Related Items

See the `viEnableEvent()`, `viWrite()`, `viTerminate()`, and `viReadAsync()` descriptions in this chapter. See the `VI_EVENT_IO_COMPLETION` description in Chapter 4, *Events*. Also see the *INSTR Resource* description in Appendix C, *Resources*.

A

Data Types

This appendix lists and describes the type assignments for ANSI C and Visual Basic for each VISA data type.

Table A-1. Type Assignments

VISA Data Type	ANSI C Binding	Visual Basic Binding	Description
ViUInt32	unsigned long	Long	A 32-bit unsigned integer.
ViPUInt32	ViUInt32 *	N/A	The location of a 32-bit unsigned integer.
ViAUInt32	ViUInt32[]	N/A	An array of 32-bit unsigned integers.
ViInt32	signed long	Long	A 32-bit signed integer.
ViPInt32	ViInt32 *	N/A	The location of a 32-bit signed integer.
ViAInt32	ViInt32[]	N/A	An array of 32-bit signed integers.
ViUInt16	unsigned short	Integer	A 16-bit unsigned integer.
ViPUInt16	ViUInt16 *	N/A	The location of a 16-bit unsigned integer.
ViAUInt16	ViUInt16[]	N/A	An array of 16-bit unsigned integers.
ViInt16	signed short	Integer	A 16-bit signed integer.
ViPInt16	ViInt16 *	N/A	The location of a 16-bit signed integer.
ViAInt16	ViInt16[]	N/A	An array of 16-bit signed integers.
ViUInt8	unsigned char	Byte	An 8-bit unsigned integer.
ViPUInt8	ViUInt8 *	N/A	The location of an 8-bit unsigned integer.
ViAUInt8	ViUInt8[]	N/A	An array of 8-bit unsigned integers.
ViInt8	signed char	Byte	An 8-bit signed integer.
ViPInt8	ViInt8 *	N/A	The location of an 8-bit signed integer.
ViAInt8	ViInt8[]	N/A	An array of 8-bit signed integers.

Table A-1. Type Assignments (Continued)

VISA Data Type	ANSI C Binding	Visual Basic Binding	Description
ViAddr	void *	Long	A type that references another data type, in cases where the other data type may vary depending on a particular context.
ViPAddr	ViAddr *	N/A	The location of a ViAddr.
ViAAddr	ViAddr[]	N/A	An array of type ViAddr.
ViChar	char	Byte	An 8-bit integer representing an ASCII character.
ViPChar	ViChar *	N/A	The location of a ViChar.
ViAChar	ViChar[]	N/A	An array of type ViChar.
ViByte	unsigned char	Byte	An 8-bit unsigned integer representing an extended ASCII character.
ViPByte	ViByte *	N/A	The location of a ViByte.
ViAByte	ViByte[]	N/A	An array of type ViByte.
ViBoolean	ViUInt16	Integer	A type for which there are exactly two complementary values: VI_TRUE and VI_FALSE.
ViPBoolean	ViBoolean *	N/A	The location of a ViBoolean.
ViABoolean	ViBoolean[]	N/A	An array of type ViBoolean.
ViReal32	float	Single	A 32-bit single-precision value.
ViPReal32	ViReal32 *	N/A	The location of a 32-bit single-precision value.
ViAReal32	ViReal32[]	N/A	An array of 32-bit single-precision values.
ViReal64	double	Double	A 64-bit double-precision value.
ViPReal64	ViReal64 *	N/A	The location of a 64-bit double-precision value.
ViAReal64	ViReal64[]	N/A	An array of 64-bit double-precision values.

Table A-1. Type Assignments (Continued)

VISA Data Type	ANSI C Binding	Visual Basic Binding	Description
ViBuf	ViPByte	String	The location of a block of data.
ViPBuf	ViPByte	String	The location to store a block of data.
ViABuf	ViBuf[]	N/A	An array of type ViBuf.
ViString	ViPChar	String	The location of a NULL-terminated ASCII string.
ViPString	ViPChar	String	The location to store a NULL-terminated ASCII string.
ViAString	ViString[]	N/A	An array of type ViString.
ViRsrc	ViString	String	A ViString type that is further restricted to adhere to the addressing grammar for resources as shown in the description of the VI_ATTR_RSRC_NAME attribute in Chapter 3, *Attributes*.
ViPRsrc	ViString	String	The location to store a ViRsrc.
ViARsrc	ViRsrc[]	N/A	An array of type ViRsrc.
ViStatus	ViInt32	Long	A defined type that contains values corresponding to VISA-defined Completion and Error termination codes.
ViPStatus	ViStatus *	N/A	The location of a ViStatus.
ViAStatus	ViStatus[]	N/A	An array of type ViStatus.
ViVersion	ViUInt32	Long	A defined type that contains a reference to all information necessary for the architect to represent the current version of a resource. The most significant 12 bits contain the major revision number, the next 12 bits contain the minor revision number, and the least significant 8 bits contain the subminor revision number.
ViPVersion	ViVersion *	N/A	The location of a ViVersion.

Table A-1. Type Assignments (Continued)

VISA Data Type	ANSI C Binding	Visual Basic Binding	Description
ViAVersion	ViVersion[]	N/A	An array of type ViVersion.
ViObject	ViUInt32	Long	The most fundamental VISA data type. It contains attributes and can be closed when no longer needed.
ViPObject	ViObject *	N/A	The location of a ViObject.
ViAObject	ViObject[]	N/A	An array of type ViObject.
ViSession	ViObject	Long	A defined type that contains a reference to all information necessary for the architect to manage a communication channel with a resource.
ViPSession	ViSession *	N/A	The location of a ViSession.
ViASession	ViSession[]	N/A	An array of type ViSession.
ViAccessMode	ViUInt32	Long	A defined type that specifies the different mechanisms that control access to a resource.
ViPAccessMode	ViAccessMode *	N/A	The location of a ViAccessMode.
ViBusAddress	ViUInt32	Long	A type that represents the system-dependent physical address.
ViPBusAddress	ViBusAddress *	N/A	The location of a ViBusAddress.
ViBusSize	ViUInt32	Long	A type that represents the system-dependent physical address size.
ViAttr	ViUInt32	Long	A type that uniquely identifies an attribute.
ViAttrState	ViUInt32	Long	A value unique to the individual type of an attribute.
ViPAttrState	void *	Any	The location of a ViAttrState.
ViVAList	va_list	Any	The location of a list of a variable number of parameters of differing types.

Table A-1. Type Assignments (Continued)

VISA Data Type	ANSI C Binding	Visual Basic Binding	Description
ViEventType	ViUInt32	Long	A defined type that uniquely identifies the type of an event.
ViPEventType	ViEventType *	N/A	The location of a ViEventType.
ViEventFilter	ViUInt32	Long	A defined type that specifies filtering masks or other information unique to an event.
ViFindList	ViObject	Long	A defined type that contains a reference to all resources found during a search operation.
ViPFindList	ViFindList *	N/A	The location of a ViFindList.
ViEvent	ViObject	Long	A defined type that encapsulates the information necessary to process an event.
ViPEvent	ViEvent *	N/A	The location of a ViEvent.
ViKeyId	ViString	String	A defined type that contains a reference to all information necessary for the architect to manage the association of a thread or process and session with a lock on a resource.
ViPKeyId	ViPString	String	The location of a ViKeyId.
ViJobId	ViUInt32	N/A	A defined type that contains a reference to all information necessary for the architect to encapsulate the information necessary for a posted operation request.
ViPJobId	ViJobId *	N/A	The location of a ViJobId.
ViHndlr	ViStatus (*) (ViSession, ViEventType, ViEvent, ViAddr)	N/A	A value representing an entry point to an operation for use as a callback.

Note ***If you are using Visual Basic version 3 instead of version 4, the*** `Byte` ***data type is not available. For input parameters you use an*** `Integer` ***variable, and for output parameters you use a*** `String *1` ***variable. This is due to an incompatibility between the two versions of Visual Basic.***

B

Status Codes

This appendix lists and describes the completion and error codes.

Table B-1. Completion Codes

Completion Codes	Values	Meaning
VI_SUCCESS	0	Operation completed successfully.
VI_SUCCESS_EVENT_EN	3FFF0002h	Specified event is already enabled for at least one of the specified mechanisms.
VI_SUCCESS_EVENT_DIS	3FFF0003h	Specified event is already disabled for at least one of the specified mechanisms.
VI_SUCCESS_QUEUE_EMPTY	3FFF0004h	Operation completed successfully, but queue was already empty.
VI_SUCCESS_TERM_CHAR	3FFF0005h	The specified termination character was read.
VI_SUCCESS_MAX_CNT	3FFF0006h	The number of bytes read is equal to the input count.
VI_WARN_CONFIG_NLOADED	3FFF0077h	The specified configuration either does not exist or could not be loaded; using VISA-specified defaults.
VI_SUCCESS_DEV_NPRESENT	3FFF007Dh	Session opened successfully, but the device at the specified address is not responding.
VI_SUCCESS_QUEUE_NEMPTY	3FFF0080h	Wait terminated successfully on receipt of an event notification. There is still at least one more event occurrence of the requested type(s) available for this session.
VI_WARN_NULL_OBJECT	3FFF0082h	The specified object reference is uninitialized.

Table B-1. Completion Codes (Continued)

Completion Codes	Values	Meaning
VI_WARN_NSUP_ATTR_STATE	3FFF0084h	Although the specified state of the attribute is valid, it is not supported by this resource implementation.
VI_WARN_UNKNOWN_STATUS	3FFF0085h	The status code passed to the operation could not be interpreted.
VI_WARN_NSUP_BUF	3FFF0088h	The specified buffer is not supported.
VI_SUCCESS_NCHAIN	3FFF0098h	Event handled successfully. Do not invoke any other handlers on this session for this event.
VI_SUCCESS_NESTED_SHARED	3FFF0099h	Operation completed successfully, and this session has nested shared locks.
VI_SUCCESS_NESTED_EXCLUSIVE	3FFF009Ah	Operation completed successfully, and this session has nested exclusive locks.
VI_SUCCESS_SYNC	3FFF009Bh	Asynchronous operation request was actually performed synchronously.

Table B-2. Error Codes

Error Codes	Values	Meaning
VI_ERROR_SYSTEM_ERROR	BFFF0000h	Unknown system error (miscellaneous error).
VI_ERROR_INV_OBJECT	BFFF000Eh	The given session or object reference is invalid.
VI_ERROR_RSRC_LOCKED	BFFF000Fh	Specified type of lock cannot be obtained or specified operation cannot be performed, because the resource is locked.
VI_ERROR_INV_EXPR	BFFF0010h	Invalid expression specified for search.
VI_ERROR_RSRC_NFOUND	BFFF0011h	Insufficient location information or the device or resource is not present in the system.

Table B-2. Error Codes (Continued)

Error Codes	Values	Meaning
VI_ERROR_INV_RSRC_NAME	BFFF0012h	Invalid resource reference specified. Parsing error.
VI_ERROR_INV_ACC_MODE	BFFF0013h	Invalid access mode.
VI_ERROR_TMO	BFFF0015h	Timeout expired before operation completed.
VI_ERROR_CLOSING_FAILED	BFFF0016h	Unable to deallocate the previously allocated data structures corresponding to this session or object reference.
VI_ERROR_INV_DEGREE	BFFF001Bh	Specified degree is invalid.
VI_ERROR_INV_JOB_ID	BFFF001Ch	Specified job identifier is invalid.
VI_ERROR_NSUP_ATTR	BFFF001Dh	The specified attribute is not defined or supported by the referenced session, event, or find list.
VI_ERROR_NSUP_ATTR_STATE	BFFF001Eh	The specified state of the attribute is not valid, or is not supported as defined by the session, event, or find list.
VI_ERROR_ATTR_READONLY	BFFF001Fh	The specified attribute is read-only.
VI_ERROR_INV_LOCK_TYPE	BFFF0020h	The specified type of lock is not supported by this resource.
VI_ERROR_INV_ACCESS_KEY	BFFF0021h	The access key to the resource associated with this session is invalid.
VI_ERROR_INV_EVENT	BFFF0026h	Specified event type is not supported by the resource.
VI_ERROR_INV_MECH	BFFF0027h	Invalid mechanism specified.
VI_ERROR_HNDLR_NINSTALLED	BFFF0028h	A handler is not currently installed for the specified event.

Table B-2. Error Codes (Continued)

Error Codes	Values	Meaning
VI_ERROR_INV_HNDLR_REF	BFFF0029h	The given handler reference is invalid.
VI_ERROR_INV_CONTEXT	BFFF002Ah	Specified event context is invalid.
VI_ERROR_NENABLED	BFFF002Fh	The session must be enabled for events of the specified type in order to receive them.
VI_ERROR_ABORT	BFFF0030h	The operation was aborted.
VI_ERROR_RAW_WR_PROT_VIOL	BFFF0034h	Violation of raw write protocol occurred during transfer.
VI_ERROR_RAW_RD_PROT_VIOL	BFFF0035h	Violation of raw read protocol occurred during transfer.
VI_ERROR_OUTP_PROT_VIOL	BFFF0036h	Device reported an output protocol error during transfer.
VI_ERROR_INP_PROT_VIOL	BFFF0037h	Device reported an input protocol error during transfer.
VI_ERROR_BERR	BFFF0038h	Bus error occurred during transfer.
VI_ERROR_INV_SETUP	BFFF003Ah	Unable to start operation because setup is invalid (due to attributes being set to an inconsistent state).
VI_ERROR_QUEUE_ERROR	BFFF003Bh	Unable to queue asynchronous operation.
VI_ERROR_ALLOC	BFFF003Ch	Insufficient system resources to perform necessary memory allocation.
VI_ERROR_INV_MASK	BFFF003Dh	Invalid buffer mask specified.
VI_ERROR_IO	BFFF003Eh	Could not perform operation because of I/O error.
VI_ERROR_INV_FMT	BFFF003Fh	A format specifier in the format string is invalid.

Table B-2. Error Codes (Continued)

Error Codes	Values	Meaning
VI_ERROR_NSUP_FMT	BFFF0041h	A format specifier in the format string is not supported.
VI_ERROR_LINE_IN_USE	BFFF0042h	The specified trigger line is currently in use.
VI_ERROR_SRQ_NOCCURRED	BFFF004Ah	Service request has not been received for the session.
VI_ERROR_INV_SPACE	BFFF004Eh	Invalid address space specified.
VI_ERROR_INV_OFFSET	BFFF0051h	Invalid offset specified.
VI_ERROR_INV_WIDTH	BFFF0052h	Invalid source or destination width specified.
VI_ERROR_NSUP_OFFSET	BFFF0054h	Specified offset is not accessible from this hardware.
VI_ERROR_NSUP_VAR_WIDTH	BFFF0055h	Cannot support source and destination widths that are different.
VI_ERROR_WINDOW_NMAPPED	BFFF0057h	The specified session is not currently mapped.
VI_ERROR_RESP_PENDING	BFFF0059h	A previous response is still pending, causing a multiple query error.
VI_ERROR_NLISTENERS	BFFF005Fh	No Listeners condition is detected (both NRFD and NDAC are deasserted).
VI_ERROR_NCIC	BFFF0060h	The interface associated with this session is not currently the controller in charge.
VI_ERROR_NSYS_CNTLR	BFFF0061h	The interface associated with this session is not the system controller.
VI_ERROR_NSUP_OPER	BFFF0067h	The given session or object reference does not support this operation.
VI_ERROR_ASRL_PARITY	BFFF006Ah	A parity error occurred during transfer.

Table B-2. Error Codes (Continued)

Error Codes	Values	Meaning
VI_ERROR_ASRL_FRAMING	BFFF006Bh	A framing error occurred during transfer.
VI_ERROR_ASRL_OVERRUN	BFFF006Ch	An overrun error occurred during transfer. A character was not read from the hardware before the next character arrived.
VI_ERROR_NSUP_ALIGN_OFFSET	BFFF0070h	The specified offset is not properly aligned for the access width of the operation.
VI_ERROR_USER_BUF	BFFF0071h	A specified user buffer is not valid or cannot be accessed for the required size.
VI_ERROR_RSRC_BUSY	BFFF0072h	The resource is valid, but VISA cannot currently access it.
VI_ERROR_NSUP_WIDTH	BFFF0076h	Specified width is not supported by this hardware.
VI_ERROR_INV_PARAMETER	BFFF0078h	The value of some parameter—which parameter is not known—is invalid.
VI_ERROR_INV_PROT	BFFF0079h	The protocol specified is invalid.
VI_ERROR_INV_SIZE	BFFF007Bh	Invalid size of window specified.
VI_ERROR_WINDOW_MAPPED	BFFF0080h	The specified session currently contains a mapped window.
VI_ERROR_NIMPL_OPER	BFFF0081h	The given operation is not implemented.
VI_ERROR_INV_LENGTH	BFFF0083h	Invalid length specified.
VI_ERROR_INV_MODE	BFFF0091h	The specified mode is invalid.
VI_ERROR_SESN_NLOCKED	BFFF009Ch	The current session did not have any lock on the resource.

Table B-2. Error Codes (Continued)

Error Codes	Values	Meaning
VI_ERROR_MEM_NSHARED	BFFF009Dh	The device does not export any memory.
VI_ERROR_LIBRARY_NFOUND	BFFF009Eh	A code library required by VISA could not be located or loaded.

C

Resources

This appendix lists the attributes, events, and operations in each resource in VISA. Refer to Chapter 3, *Attributes*, Chapter 4, *Events*, and Chapter 5, *Operations*, for more details.

VISA Resource Template

This section lists the attributes, events, and operations for the VISA Resource Template. The attributes, events, and operations in the VISA Resource Template are available to all other resources.

Attributes

```
VI_ATTR_MAX_QUEUE_LENGTH
VI_ATTR_RM_SESSION
VI_ATTR_RSRC_IMPL_VERSION
VI_ATTR_RSRC_LOCK_STATE
VI_ATTR_RSRC_MANF_ID
VI_ATTR_RSRC_MANF_NAME
VI_ATTR_RSRC_NAME
VI_ATTR_RSRC_SPEC_VERSION
VI_ATTR_USER_DATA
```

Events

```
VI_EVENT_EXCEPTION
```

Operations

```
viClose(vi)
viDisableEvent(vi, eventType, mechanism)
viDiscardEvents(vi, eventType, mechanism)
viEnableEvent(vi, eventType, mechanism, context)
viGetAttribute(vi, attribute, attrState)
viInstallHandler(vi, eventType, handler,
                 userHandle)
viLock(vi, lockType, timeout, requestedKey,
       accessKey)
viSetAttribute(vi, attribute, attrState)
```

```
viStatusDesc(vi, status, desc)
viTerminate(vi, degree, jobId)
viUninstallHandler(vi, eventType, handler,
                   userHandle)
viUnlock(vi)
viWaitOnEvent(vi, inEventType, timeout,
              outEventType, outContext)
```

VISA Resource Manager

This section lists the attributes, events, and operations for the VISA Resource Manager. The attributes, events, and operations in the VISA Resource Template are available to this resource in addition to the operations listed below.

Attributes

The attributes for the VISA Resource Template are available to this resource. This resource has no defined attributes of its own.

Events

None

Operations

```
viFindNext(findList, instrDesc)
viFindRsrc(sesn, expr, findList, retcnt,
           instrDesc)
viOpen(sesn, rsrcName, accessMode, timeout, vi)
viOpenDefaultRM(sesn)
```

INSTR Resource

This section lists the attributes, events, and operations for the INSTR Resource. The attributes, events, and operations in the VISA Resource Template are available to this resource in addition to the attributes and operations listed below.

Attributes

VI_ATTR_ASRL_AVAIL_NUM
VI_ATTR_ASRL_BAUD
VI_ATTR_ASRL_CTS_STATE
VI_ATTR_ASRL_DATA_BITS
VI_ATTR_ASRL_DCD_STATE
VI_ATTR_ASRL_DSR_STATE
VI_ATTR_ASRL_DTR_STATE
VI_ATTR_ASRL_END_IN
VI_ATTR_ASRL_END_OUT
VI_ATTR_ASRL_FLOW_CONTROL
VI_ATTR_ASRL_PARITY
VI_ATTR_ASRL_REPLACE_CHAR
VI_ATTR_ASRL_RI_STATE
VI_ATTR_ASRL_RTS_STATE
VI_ATTR_ASRL_STOP_BITS
VI_ATTR_ASRL_XOFF_CHAR
VI_ATTR_ASRL_XON_CHAR
VI_ATTR_CMDR_LA
VI_ATTR_DEST_ACCESS_PRIV
VI_ATTR_DEST_BYTE_ORDER
VI_ATTR_DEST_INCREMENT
VI_ATTR_FDC_CHNL
VI_ATTR_FDC_GEN_SIGNAL_EN
VI_ATTR_FDC_MODE
VI_ATTR_FDC_USE_PAIR
VI_ATTR_GPIB_PRIMARY_ADDR
VI_ATTR_GPIB_READDR_EN
VI_ATTR_GPIB_REN_STATE
VI_ATTR_GPIB_SECONDARY_ADDR
VI_ATTR_GPIB_UNADDR_EN
VI_ATTR_IMMEDIATE_SERV
VI_ATTR_INTF_INST_NAME
VI_ATTR_INTF_NUM
VI_ATTR_INTF_PARENT_NUM
VI_ATTR_INTF_TYPE

VI_ATTR_IO_PROT
VI_ATTR_MAINFRAME_LA
VI_ATTR_MANF_ID
VI_ATTR_MEM_BASE
VI_ATTR_MEM_SIZE
VI_ATTR_MEM_SPACE
VI_ATTR_MODEL_CODE
VI_ATTR_RD_BUF_OPER_MODE
VI_ATTR_SEND_END_EN
VI_ATTR_SLOT
VI_ATTR_SRC_ACCESS_PRIV
VI_ATTR_SRC_BYTE_ORDER
VI_ATTR_SRC_INCREMENT
VI_ATTR_SUPPRESS_END_EN
VI_ATTR_TERMCHAR
VI_ATTR_TERMCHAR_EN
VI_ATTR_TMO_VALUE
VI_ATTR_TRIG_ID
VI_ATTR_VXI_LA
VI_ATTR_WIN_ACCESS
VI_ATTR_WIN_ACCESS_PRIV
VI_ATTR_WIN_BASE_ADDR
VI_ATTR_WIN_BYTE_ORDER
VI_ATTR_WIN_SIZE
VI_ATTR_WR_BUF_OPER_MODE

Events

VI_EVENT_IO_COMPLETION
VI_EVENT_SERVICE_REQ
VI_EVENT_TRIG
VI_EVENT_VXI_SIGP
VI_EVENT_VXI_VME_INTR

Operations

viAssertTrigger(vi, protocol)
viBufRead(vi, buf, count, retCount)
viBufWrite(vi, buf, count, retCount)
viClear(vi)
viFlush(vi, mask)
viGpibControlREN(vi, mode)
viIn8(vi, space, offset, val8)
viIn16(vi, space, offset, val16)

```
viIn32(vi, space, offset, val32)
viMapAddress(vi, mapSpace, mapBase, mapSize,
access, suggested, address)
viMemAlloc(vi, size, offset)
viMemFree(vi, offset)
viMove(vi, srcSpace, srcOffset, srcWidth,
        destSpace, destOffset, destWidth, length)
viMoveAsync(vi, srcSpace, srcOffset, srcWidth,
             destSpace, destOffset, destWidth,
             length, jobId)
viMoveIn8(vi, space, offset, length, buf8)
viMoveIn16(vi, space, offset, length, buf16)
viMoveIn32(vi, space, offset, length, buf32)
viMoveOut8(vi, space, offset, length, buf8)
viMoveOut16(vi, space, offset, length, buf16)
viMoveOut32(vi, space, offset, length, buf32)
viOut8(vi, space, offset, val8)
viOut16(vi, space, offset, val16)
viOut32(vi, space, offset, val32)
viPeek8(vi, addr, val8)
viPeek16(vi, addr, val16)
viPeek32(vi, addr, val32)
viPoke8(vi, addr, val8)
viPoke16(vi, addr, val16)
viPoke32(vi, addr, val32)
viPrintf(vi, writeFmt, ...)
viQueryf(vi, writeFmt, readFmt, ...)
viRead(vi, buf, count, retCount)
viReadAsync(vi, buf, count, jobId)
viReadSTB(vi, status)
viScanf(vi, readFmt, ...)
viSetBuf(vi, mask, size)
viSPrintf(vi, buf, writeFmt, ...)
viSScanf(vi, buf, readFmt, ...)
viUnmapAddress(vi)
viVPrintf(vi, writeFmt, params)
viVQueryf(vi, writeFmt, readFmt, params)
viVScanf(vi, readFmt, params)
viVSPrintf(vi, buf, writeFmt, params)
viVSScanf(vi, buf, readFmt, params)
viVxiCommandQuery(vi, mode, cmd, response)
viWrite(vi, buf, count, retCount)
viWriteAsync(vi, buf, count, jobId)
```

MEMACC Resource

This section lists the attributes, events, and operations for the MEMACC Resource. The attributes, events, and operations in the VISA Resource Template are available to this resource in addition to the attributes and operations listed below.

Attributes

VI_ATTR_DEST_ACCESS_PRIV
VI_ATTR_DEST_BYTE_ORDER
VI_ATTR_DEST_INCREMENT
VI_ATTR_GPIB_PRIMARY_ADDR
VI_ATTR_GPIB_SECONDARY_ADDR
VI_ATTR_INTF_INST_NAME
VI_ATTR_INTF_NUM
VI_ATTR_INTF_PARENT_NUM
VI_ATTR_INTF_TYPE
VI_ATTR_SRC_ACCESS_PRIV
VI_ATTR_SRC_BYTE_ORDER
VI_ATTR_SRC_INCREMENT
VI_ATTR_TMO_VALUE
VI_ATTR_VXI_LA
VI_ATTR_WIN_ACCESS
VI_ATTR_WIN_ACCESS_PRIV
VI_ATTR_WIN_BASE_ADDR
VI_ATTR_WIN_BYTE_ORDER
VI_ATTR_WIN_SIZE

Events

VI_EVENT_IO_COMPLETION

Operations

```
viIn8(vi, space, offset, val8)
viIn16(vi, space, offset, val16)
viIn32(vi, space, offset, val32)
viMapAddress(vi, mapSpace, mapBase, mapSize,
             access, suggested, address)
viMove(vi, srcSpace, srcOffset, srcWidth,
       destSpace, destOffset, destWidth,
       length)
```

```
viMoveAsync(vi, srcSpace, srcOffset, srcWidth,
            destSpace, destOffset, destWidth,
            length, jobId)
viMoveIn8(vi, space, offset, length, buf8)
viMoveIn16(vi, space, offset, length, buf16)
viMoveIn32(vi, space, offset, length, buf32)
viMoveOut8(vi, space, offset, length, buf8)
viMoveOut16(vi, space, offset, length, buf16)
viMoveOut32(vi, space, offset, length, buf32)
viOut8(vi, space, offset, val8)
viOut16(vi, space, offset, val16)
viOut32(vi, space, offset, val32)
viPeek8(vi, addr, val8)
viPeek16(vi, addr, val16)
viPeek32(vi, addr, val32)
viPoke8(vi, addr, val8)
viPoke16(vi, addr, val16)
viPoke32(vi, addr, val32)
viUnmapAddress(vi)
```

D

Customer Communication

For your convenience, this appendix contains forms to help you gather the information necessary to help us solve your technical problems and a form you can use to comment on the product documentation. When you contact us, we need the information on the Technical Support Form and the configuration form, if your manual contains one, about your system configuration to answer your questions as quickly as possible.

National Instruments has technical assistance through electronic, fax, and telephone systems to quickly provide the information you need. Our electronic services include a bulletin board service, an FTP site, a fax-on-demand system, and e-mail support. If you have a hardware or software problem, first try the electronic support systems. If the information available on these systems does not answer your questions, we offer fax and telephone support through our technical support centers, which are staffed by applications engineers.

Electronic Services

Bulletin Board Support

National Instruments has BBS and FTP sites dedicated for 24-hour support with a collection of files and documents to answer most common customer questions. From these sites, you can also download the latest instrument drivers, updates, and example programs. For recorded instructions on how to use the bulletin board and FTP services and for BBS automated information, call 512 795 6990. You can access these services at:

United States: 512 794 5422
Up to 14,400 baud, 8 data bits, 1 stop bit, no parity

United Kingdom: 01635 551422
Up to 9,600 baud, 8 data bits, 1 stop bit, no parity

France: 01 48 65 15 59
Up to 9,600 baud, 8 data bits, 1 stop bit, no parity

FTP Support

To access our FTP site, log on to our Internet host, `ftp.natinst.com`, as `anonymous` and use your Internet address, such as `joesmith@anywhere.com`, as your password. The support files and documents are located in the `/support` directories.

Fax-on-Demand Support

Fax-on-Demand is a 24-hour information retrieval system containing a library of documents on a wide range of technical information. You can access Fax-on-Demand from a touch-tone telephone at 512 418 1111.

E-Mail Support (Currently USA Only)

You can submit technical support questions to the applications engineering team through e-mail at the Internet address listed below. Remember to include your name, address, and phone number so we can contact you with solutions and suggestions.

`support@natinst.com`

Telephone and Fax Support

National Instruments has branch offices all over the world. Use the list below to find the technical support number for your country. If there is no National Instruments office in your country, contact the source from which you purchased your software to obtain support.

Country	Telephone	Fax
Australia	03 9879 5166	03 9879 6277
Austria	0662 45 79 90 0	0662 45 79 90 19
Belgium	02 757 00 20	02 757 03 11
Brazil	011 288 3336	011 288 8528
Canada (Ontario)	905 785 0085	905 785 0086
Canada (Quebec)	514 694 8521	514 694 4399
Denmark	45 76 26 00	45 76 26 02
Finland	09 725 725 11	09 725 725 55
France	01 48 14 24 24	01 48 14 24 14
Germany	089 741 31 30	089 714 60 35
Hong Kong	2645 3186	2686 8505
Israel	03 6120092	03 6120095
Italy	02 413091	02 41309215
Japan	03 5472 2970	03 5472 2977
Korea	02 596 7456	02 596 7455
Mexico	5 520 2635	5 520 3282
Netherlands	0348 433466	0348 430673
Norway	32 84 84 00	32 84 86 00
Singapore	2265886	2265887
Spain	91 640 0085	91 640 0533
Sweden	08 730 49 70	08 730 43 70
Switzerland	056 200 51 51	056 200 51 55
Taiwan	02 377 1200	02 737 4644
United Kingdom	01635 523545	01635 523154
United States	512 795 8248	512 794 5678

Technical Support Form

Photocopy this form and update it each time you make changes to your software or hardware, and use the completed copy of this form as a reference for your current configuration. Completing this form accurately before contacting National Instruments for technical support helps our applications engineers answer your questions more efficiently.

If you are using any National Instruments hardware or software products related to this problem, include the configuration forms from their user manuals. Include additional pages if necessary.

Name _____

Company _____

Address _____

_____

Fax (___) _____Phone (___) _____

Computer brand_____ Model _____Processor_____

Operating system (include version number) _____

Clock speed _____MHz RAM _____MB Display adapter _____

Mouse ___yes ___no Other adapters installed_____

Hard disk capacity _____MB Brand_____

Instruments used _____

_____

National Instruments hardware product model _____ Revision _____

Configuration _____

National Instruments software product _____ Version _____

Configuration _____

The problem is: _____

_____

_____

_____

_____

List any error messages: _____

_____

_____

The following steps reproduce the problem: _____

_____

_____

_____

_____

Documentation Comment Form

National Instruments encourages you to comment on the documentation supplied with our products. This information helps us provide quality products to meet your needs.

Title: *NI-VISA™ Programmer Reference Manual*

Edition Date: April 1998

Part Number: 321073C-01

Please comment on the completeness, clarity, and organization of the manual.

_____

_____

_____

_____

_____

_____

_____

If you find errors in the manual, please record the page numbers and describe the errors.

_____

_____

_____

_____

_____

_____

_____

Thank you for your help.

Name _____

Title _____

Company _____

Address _____

_____

E-Mail Address _____

Phone (___) _____ Fax (___) _____

Mail to: Technical Publications
National Instruments Corporation
6504 Bridge Point Parkway
Austin, Texas 78730-5039

Fax to: Technical Publications
National Instruments Corporation
512 794 5678

Glossary

Prefix	Meanings	Value
n-	nano-	10^{-9}
µ-	micro-	10^{-6}
m-	milli-	10^{-3}
k-	kilo-	10^{3}
M-	mega-	10^{6}

A

Address — A string (or other language construct) that uniquely locates and identifies a resource. VISA defines an ASCII-based grammar that associates strings with particular physical devices and VISA resources.

API — Application Programming Interface. The direct interface that an end user sees when creating an application. In VISA, the API consists of the sum of all of the operations, attributes, and events of each of the VISA Resource Classes.

Attribute — A value within an object or resource that reflects a characteristic of its operational state.

B

b — Bit

B — Byte

Bus Error — An error that signals failed access to an address. Bus errors occur with low-level accesses to memory and usually involve hardware with bus mapping capabilities. For example, nonexistent memory, a nonexistent register, or an incorrect device access can cause a bus error.

C

Callback	Same as *Handler.* A software routine that is invoked when an asynchronous event occurs. In VISA, callbacks can be installed on any session that processes events.
Commander	A device that has the ability to control another device. This term can also denote the unique device that has sole control over another device (as with the VXI Commander/Servant hierarchy).
Communication Channel	The same as *Session*. A communication path between a software element and a resource. Every communication channel in VISA is unique.
Controller	An entity that can control another device(s) or is in the process of performing an operation on another device.

D

Device	An entity that receives commands from a controller. A device can be an instrument, a computer (acting in a non-controller role), or a peripheral (such as a plotter or printer).
DLL	Dynamic Link Library. Same as a *shared library* or *shared object.* A file containing a collection of functions that can be used by multiple applications. This term is usually used for libraries on Windows platforms.

E

Event	An asynchronous occurrence that is independent of the normal sequential execution of the process running in a system.

F

FIFO	First In-First Out; a method of data storage in which the first element stored is the first one retrieved.

H

Handler	Same as *Callback*. A software routine that is invoked when an asynchronous event occurs. In VISA, callbacks can be installed on any session that processes events.

I

Instrument	A device that accepts some form of stimulus to perform a designated task, test, or measurement function. Two common forms of stimuli are message passing and register reads and writes. Other forms include triggering or varying forms of asynchronous control.
Instrument Driver	A set of routines designed to control a specific instrument or family of instruments, and any necessary related files for LabWindows/CVI or LabVIEW.
Interface	A generic term that applies to the connection between devices and controllers. It includes the communication media and the device/controller hardware necessary for cross-communication.
Interrupt	A condition that requires attention out of the normal flow of control of a program.

L

Lock	A state that prohibits sessions other than the session(s) owning the lock from accessing a resource.

M

Mapping	An operation that returns a reference to a specified section of an address space and makes the specified range of addresses accessible to the requester. This function is independent of memory allocation.

O

Operation	An action defined by a resource that can be performed on a resource. In general, this term is synonymous with the connotation of the word *method* in object-oriented architectures.

P

Process	An operating system element that shares a system's resources. A multi-process system is a computer system that allows multiple programs to execute simultaneously, each in a separate process environment. A single-process system is a computer system that allows only a single program to execute at a given point in time.

R

Register	An address location that can be read from or written into or both. It may contain a value that is a function of the state of hardware or can be written into to cause hardware to perform a particular action. In other words, an address location that controls and/or monitors hardware.
Resource Class	The definition for how to create a particular resource. In general, this is synonymous with the connotation of the word *class* in object-oriented architectures. For VISA Instrument Control Resource Classes, this refers to the definition for how to create a resource which controls a particular capability or set of capabilities of a device.
Resource or Resource Instance	In general, this term is synonymous with the connotation of the word *object* in object-oriented architectures. For VISA, *resource* more specifically refers to a particular implementation (or *instance* in object-oriented terms) of a Resource Class.

S

s	Second
Session	The same as *Communication Channel*. A communication path between a software element and a resource. Every communication channel in VISA is unique.
Shared Library or Shared Object	Same as *DLL*. A file containing a collection of functions that can be used by multiple applications. This term is usually used for libraries on UNIX platforms.
Shared Memory	A block of memory that is accessible to both a client and a server. The memory block operates as a buffer for communication. This is unique to register-based interfaces such as VXI.

SRQ	IEEE 488 Service Request. This is an asynchronous request from a remote device that requires service. A service request is essentially an interrupt from a remote device. For GPIB, this amounts to asserting the SRQ line on the GPIB. For VXI, this amounts to sending the Request for Service True event (REQT).
Status Byte	A byte of information returned from a remote device that shows the current state and status of the device. If the device follows IEEE 488 conventions, bit 6 of the status byte indicates whether the device is currently requesting service.

T

Thread	An operating system element that consists of a flow of control within a process. In some operating systems, a single process can have multiple threads, each of which can access the same data space within the process. However, each thread has its own stack and all threads can execute concurrently with one another (either on multiple processors, or by time-sharing a single processor).

V

Virtual Instrument	A name given to the grouping of software modules (in this case, VISA resources with any associated or required hardware) to give the functionality of a traditional stand-alone instrument. Within VISA, a virtual instrument is the logical grouping of any of the VISA resources.
VISA	Virtual Instrument Software Architecture. This is the general name given to this product and its associated architecture. The architecture consists of two main VISA components: the VISA Resource Manager and the VISA Instrument Control Resources.
VISA Instrument Control Resources	This is the name given to the part of VISA that defines all of the device-specific resource classes. VISA Instrument Control Resources encompass all defined device capabilities for direct, low-level instrument control.
VISA Resource Manager	This is the name given to the part of VISA that manages resources. This management includes support for finding and opening resources.

VISA Resource Template	This is the name given to the part of VISA defines the basic constraints and interface definition for the creation and use of a VISA resource. All VISA resources must derive their interface from the definition of the VISA Resource Template. This includes services for setting and retrieving attributes, receiving events, locking resources, and closing objects.

Index

Numbers

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X